Universidad Zaragoza
1542

Trabajo Fin de Máster

Reducción de dimensionalidad y técnicas de inferencia de estado para sensores inteligentes

Dimensionality reduction and simple status inference techniques for smart embedded sensors

Autor

Jorge Vizárraga Turmo

Director/es

Julio David Buldaín Pérez
Roberto Casas Nebra

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2019

RESUMEN

REDUCCIÓN DE DIMENSIONALIDAD Y TÉCNICAS DE INFERENCIA DE ESTADO PARA SENSORES INTELIGENTES

Este proyecto nace del intento de reducir la cantidad de datos enviados por sensores inteligentes para prolongar así su vida útil y dotarles de cierta seguridad al realizar una abstracción de los datos en crudo haciendo imposible su interpretación.

Para ello se hace uso de una técnica ya desarrollada hace años, el vector quantization pero añadiéndole una serie de mejoras que permitan recalcular la disposición de los centroides para conseguir minimizar el error de reconstrucción.
Se ha desarrollado esta técnica en lenguaje Python y se va a evaluar con una base de datos existente variando, los parámetros de muestreo, el uso de codificación por media y desviación típica, la composición con una o las tres componentes del sensor inercial y el tamaño de la red de entrenamiento.

Con este sistema definido, se propone una clasificación de los datos reducidos para intentar sacar aun mayor índice de compresión frente a los datos enviados por el sensor en crudo.

Una vez verificado el comportamiento del modelo se implementa en lenguaje micro Python en un sensor inteligente para evaluar el sistema compresor frente a su uso convencional.

Contenido

Capítulo 1 Introducción

1.1. Motivación

Este trabajo Fin de Máster (TFM) se ha desarrollado dentro del grupo de Investigación HOWLab (Human Openware Research Lab), cuyo objetivo principal es la investigación y el desarrollo de tecnologías centradas en las personas y sus entornos.

La reducción de dimensionalidad para envió de datos es un tema que preocupa cada vez más, ya que con la gran evolución en el campo de los sensores como el *IoT*, la industria 4.0, *Smart-City*, está generando unas cantidades enormes de datos. De ahí que conseguir reducir la dimensión de los datos enviados permita garantizar una mayor autonomía energética del dispositivo al realizar menos envíos, ya que esto suele ser la parte más costosa del proceso de sensado. Además de introducir una abstracción a los datos que impidan ser interpretados sin la clave por parte de terceros y reducir el nivel de flujo de datos por el canal de envío.

1.2. Objetivos

El objetivo de este proyecto es conseguir un método de compresión eficiente para datos en sensores inteligentes, en este caso se centra en los sensores de tipo inercial.

Este método debe conseguir que el sistema reduzca el *datarate* de envío de datos, su consumo energético permitiendo asegurar una mayor autonomía y un añadido de cifrado a los datos al incorporarles una abstracción mediante el modelo compresor y la distorsión por medio de sus parámetros estadísticos.

Se deberá plantear el modelo compresor estudiando los existentes, realizar un algoritmo en la plataforma de desarrollo, parametrizarlo y evaluarlo.

Añadirle una funcionalidad de inferencia de estado para minimizar aún más la cantidad de datos enviados y permitir personalizar cada tabla según el usuario.

Ver el añadido de compresión respecto a los métodos tradicionales.

El objetivo final será la incorporación en un sensor inteligente para evaluación real del método compresor.

1.3. Metodología

De esta manera, los pasos a seguir para el cumplimiento de los objetivos planteados son los siguientes:

- Estudio del estado del arte y aprendizaje sobre el uso las distintas técnicas de reducción de dimensionalidad y pre-procesado.
- Búsqueda de posibles bases de datos para evaluación offline del modelo compresor.
- Análisis Base de datos escogida.
- Desarrollo del modelo compresor VQ adaptativo en Python.
- Extracción de los parámetros de entrenamiento.
- Entrenamiento del modelo adaptativo.
- Análisis resultados compresión.
- Evaluación Inferencia de estado
- Implementación hardware
- Migración código MicroPython

1.4. Herramientas utilizadas

Para el desarrollo del proyecto, se ha hecho uso de las siguientes herramientas:

- PYCHARM 2019 2.3: para programación en Python, donde se han desarrollado los scripts con las librerías desarrolladas en el proyecto y las de las librerías: scikit-learn[1], pandas[2] y numpy [3].
- Microsoft Visio 2019: para desarrollar los diagramas de flujo del proyecto.
- Visual Studio Code con el plugin Pymark: para programación en MicroPython, donde se ha llevado a cabo la programación del sensor.
- Google Colaboratory: plataforma que te permite de manera gratuita un entorno de ejecución virtual, con 12GB de RAM y una tarjeta gráfica Tesla K80 GPU. Esta plataforma se ha utilizado junto al PC para el entrenamiento las redes y evaluación de las bases de Datos.
- Gestor de referencias RefWorks: para añadir las citas en el documento Word.

1.5. Estructura de la memoria

La presente memoria recoge el trabajo realizado durante el desarrollo de este TFM. Está estructurada en los siguientes capítulos:

- Capítulo 1: Introduce el proyecto, se presenta el marco del proyecto, se expone los objetivos y la metodología a seguir para el desarrollo de este.
- Capítulo 2: Muestra el estado de las técnicas actuales de compresión de datos, así como clasificación, junto a este se incorpora la búsqueda de bases de datos.
- Capítulo 3: Define el modelo de compresión *vector quantization (VQ),* el pre-procesado de los datos de entrada, así como la interpretación adaptativa que se le da este proyecto y un añadido para realizar una clasificación en función del estado.
- Capítulo 4: Concluye con el análisis de los parámetros del modelo compresor y sus resultados.
- Capítulo 5: Implementación hardware.
- Capítulo 6: Conclusiones.
- Referencias.
- Anexos.

Capítulo 2 Estado del arte

2.1. Introducción

Los sensores inteligentes están en nuestro día a día y ya es raro ver algo que no los incorpore para tomar datos de procesos, acciones o del propio entorno, no hace falta irse lejos para ver el gran impacto que tienen en la sociedad, los encontramos en casas, automóviles, industrias y en forma de complementos de tal manera que una persona pueda llevar encima (*Wearables*).

De ahí el gran interés por el gran flujo de información que se puede extraer para conseguir datos de manera automática, evitando el tedioso trabajo de formar bases de datos. Uno de los problemas que podemos encontrar muchas veces es que estos sensores no puedan tener un fácil acceso a medios de energía o que su intención no sea la de poder ser recargados. Esta dependencia energética hace que se busque un consumo mínimo para aumentar la vida útil del mismo, por eso conseguir minimizar las tareas de envío de datos es de enorme importancia.

Añadir esa compresión de datos no es tarea fácil ya que las técnicas que normalmente se usan están pensadas para ser procesadas de manera externa al sensor, ya bien en la nube o en dispositivos remotos. Es por eso por lo que incorporar nuevas técnicas para reducir la dimensionalidad es un tema tan interesante, también esto plantea una abstracción en los datos.

Con esta codificación de los datos se conseguirá que terceros no sean capaces de robar los datos que estamos transmitiendo

El sensor escogido para este caso de estudio es uno de clase inercial que incorpore como mínimo datos de aceleración y giróscopo, y que suele estar presente en técnicas HAR (Human Activity Recognition) pero también se está abriendo a otros mercados como el reconocimiento de actividades en mascotas o la ganadería.

2.2. Técnicas de reducción de dimensionalidad

El objetivo principal de este trabajo es conseguir reducir el tamaño de los datos, para llevarlo a cabo se va a realizar un estudio de las técnicas y modelos que se usan en sensores de tipo inercial como el del caso de estudio.

Para poder hablar de reducción de dimensionalidad antes hay que hablar de la adquisición de datos, en el cual va a estar el primer paso el pre-procesado de los datos en crudo, en la Figura 1 podemos ver una línea temporal de los pasos a seguir para analizar correctamente cualquier conjunto de datos.

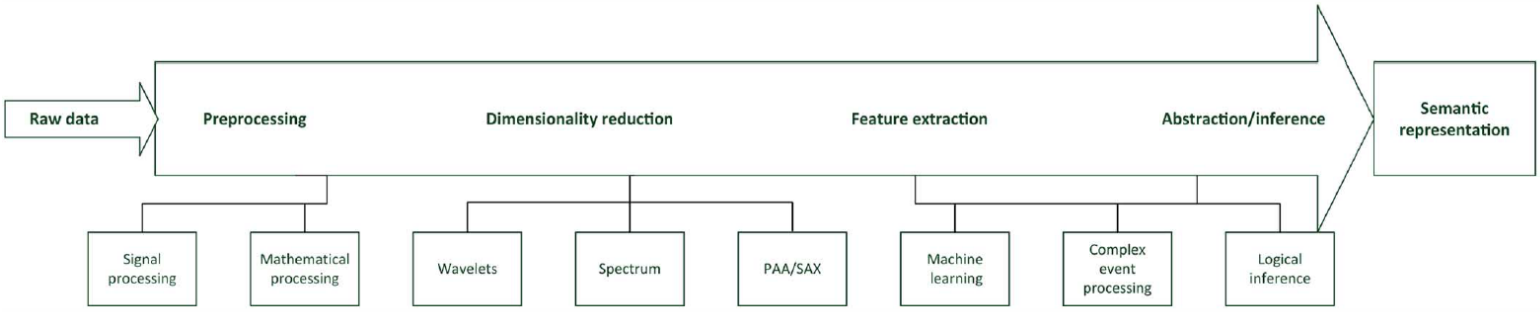


Figura 1. Proceso de abstracción de datos[4]

2.2.1. Pre-Procesado

Ante los datos de entrada del sensor se pueden distinguir varias técnicas para reducir ruidos introducidos en la medida:

- Filtros de señal

  Podemos encontrar filtros hardware o software con el objetivo de eliminar ese ruido conocido como *outliers*.
  Los más típicos para el caso de sensores inerciales [5] son el paso banda [4], si tenemos definido un intervalo de frecuencias, o un paso alto/bajo [4] si queremos cortar a partir de una frecuencia crítica.

- Métodos estadísticos

  Para diferenciar entre cambios bruscos podríamos usar el mínimo y máximo, también podemos usarlos para el escalado global de los datos, si queremos mejorar su representación usaríamos la varianza [4], [5] y desviación típica y para el caso de reducir el ruido y los picos la media y la mediana.

  Existen un gran número de características a parte de estas anteriores que se evalúan sensores inerciales en la Tabla 1 se recogen las empleadas en los estudios.

  En este tipo de datos de movimiento no es raro encontrar el uso de ventanas deslizantes [6], [7],[8] como aparece en la Figura 2 ya bien para aumentar el número de datos de entrada, como para mejorar el *tracking* de la señal y mejorar los procesos posteriores de reconstrucción y clasificación.

a b

Figura 2. a) Filtrado señal de entrada, b) Ventanas deslizantes

Autores	Features extraídos
[5]	Desviación típica, media, máximo, mínimo, suma absoluta, mediana, rms, varianza, correlación ejes XY, YZ y XZ, rango, energía.
[9]	Desviación típica, media, máximo, mínimo, suma absoluta, mediana, velocidad angular, correlación ejes XY, YZ y XZ, percentiles, entropía.
[10]	Media, varianza, correlación ejes, calculo energía.
[11]	Desviación típica, media, máximo, mínimo, suma absoluta, mediana, velocidad angular, correlación ejes XY, YZ y XZ, integral, entropía.
[12]	Media, mínimo, máximo, desviación típica.

Tabla 1. Features acelerómetros

2.2.2. Reducción de dimensionalidad

Una vez los datos han sido procesados podemos aplicar algunas de las técnicas de compresión como las que se citan en la Tabla 2.

En la tabla se registran los métodos de reducción de dimensionalidad que se usan para sensores de ámbito general, además de acelerómetros.

Autores	Entorno	Técnica compresión
[1]	General	Transformada de Fourier (DFT), Transformada de onda (DWT), Aproximación agregada por partes (PAA)
[9]	Acelerómetro	Principal Component Analysis (PCA)
[11]	Acelerómetro	Principal Component Analysis (PCA)
[12]	Acelerómetro	Principal Component Analysis (PCA)
[13]	General	Sequential Forward Selection (SFS), Random subset Feature selection (RSFS)
[14]	Datos de sistemas de control industriales	Principal Component Analysis (PCA), Kernel PCA variando las funciones de Kernel (lineal, polinómica y radial), ISOMAP
[15]	General	Principal Component Analysis (PCA), Independient Component Analysis (PCA) (ICA), I-PCA y otros métodos relacionados con la semántica LSA o WordNet
[16]	Huellas dactilares	Vector Quantization
[17]	General	FSCL
[18]	General	MSCL (unidad, Qerror, FSCL)

Tabla 2. Técnicas de compresión

2.3. Bases de datos

Para evaluar la técnica de compresión empleada y compararla con otras es necesario tener una base de datos del sensor en cuestión se ha llevado a cabo una búsqueda en el repositorio [19]con el termino de búsqueda acelerómetro.

Del resultado se han extraído las siguientes bases, relacionadas con el reconocimiento de actividades en humanos, para analizar su contenido podemos ir a la Tabla 3.

Para escoger la base vamos a seguir una configuración que cumpla las siguientes características: un sensor capaz de darnos la mayor parte de movimiento, una serie de usuarios reducida y unas actividades que no sean repetitivas o difíciles de caracterizar con un solo sensor.

Para el caso de estudio no es crítico el número de actividades, más bien lo óptimo para realizar la aproximación de inferencia de estado sería un caso en el que los patrones a reconocer sean lo más distintos posibles.

El número de sujetos nos puede dar una idea de lo diferentes que son los datos para cada usuario, ya que es imposible hablar de un sensor universal en una base de

datos con tan pocos usuarios, será interesante corroborar como cambian los modelos según el sujeto dando lugar a un modelo personalizado para cada uno de ellos.

El total de datos sí que es dato de interés puesto que cuanto mayor sea, más datos podremos usar para entrenar el modelo dejando una buena cantidad para test, también hay que tener en cuenta que ese total de datos se verá dividido por actividades, usuarios y sensores.

Autores	Titulo	Sujetos	Actividades	Sensores	Datos
[20]	Activity Recognition from Single Chest-Mounted Accelerometer Data Set	15	7	1	1926896
[21]	Daily and Sports Activities Data Set	8	19	9	6512500
[22]	Heterogeneity Activity Recognition Data Set	9	6	10	33741504
[23]	mHealth dataset	10	12	7	1215745
[24]	Realdisp	17	33	9	7355749

Tabla 3. Bases de datos repositorio UCI

Las actividades realizadas son tales como andar, correr, sentarse, agacharse, saltar, levantarse, ir en bici, tumbarse, subir o bajar escaleras, ejercicios de brazos. En tres de las bases de datos[21] [23] [24] algunos de estos movimientos se ven replicados, de manera que hay patrones difíciles de poder identificar quedándose solo con un sensor, que va a ser el caso de estudio.

La base de datos [20] refleja patrones un poco dudosos para el objetivo del proyecto como trabajando en el ordenador o hablando con alguien, ya que son actividades difíciles de categorizar.

Nos vamos a quedar con la base [22] ya que cumple los requisitos definidos anteriormente, el sensor esa ubicado a la altura de la cintura, hay una base con 9 usuarios. Con ese número de usuarios probaremos combinaciones entre ellos e intentaremos llevar a cabo la clasificación de sus actividades.

Estas actividades son reducidas y las conforman patrones como sentarse, levantarse, andar, subir escaleras, bajar escaleras e ir en bici. Los sensores inerciales provienen de una variedad de teléfonos móviles y varios *smartwatchs* para los que se realiza un análisis en el siguiente apartado Pre-Procesado.

2.4. Clasificación de estado

En lo referente a la clasificación del estado, este proyecto ha buscado comprimir lo máximo posible los datos enviados por el sensor, con el fin de reducir consumos y prolongar la vida útil del sensor, que en muchos casos es la prioridad principal.

No obstante, se ha realizado una búsqueda de los principales clasificadores de estado para los sensores inerciales que se encuentra en la Tabla 4, en este caso para reconocimiento de actividades humanas ya que es por la línea que se va a desarrollar el proyecto [18].

Autores	Algoritmos de clasificación	%Precisión Test
[5]	SVM	85.73
[8]	CNN, Random forest	92.71, 89.10
[9]	SVM, MLP, KNN	97.2, 97.2, 99.42
[10]	SVM, Adaboost, Regression	82.28, 92.81, 86.55
[12]	SVM, MLP	86.3, 96.48

Tabla 4. Técnicas de clasificación

2.5. Casos de estudio

Como se ha visto a lo largo del este capítulo el perfil principal de estudio es el reconocimiento de actividades, no obstante no tiene por qué ser solo de humanos ya se da casos en animales domésticos como perros ofreciendo una monitorización [25] o una clasificación de actividades [26].

También en el sector de la ganadería en ovejas[27] o hasta en peces y crustáceos [27], en relación a los antepenúltimos si hay una fuente de datos enorme que puede brindar de ayuda para los cuidadores.

2.6. Conclusiones

Analizando los resultados de la búsqueda, vamos a seguir una dinámica similar a la de la Figura 1, primero escogiendo una de las bases de datos que hemos recogido en la Tabla 3, esto aparece en el apartado de Pre-Procesado.

Evaluaremos en principio solo la media y la desviación típica para aplicárselas a los datos de entrada para mejorar los datos de entrada y reducir el ruido, de paso nos servirá para añadir una clave a la decodificación de la señal enviada.

De los métodos estudiados el más usado es PCA [9], [11], [12], [14], [15], será interesante compararlo con el método de estudio para el que se va a plantear este TFM un modelo Vector Quantization pero añadiendo una evaluación de magnitudes[18].

El hecho de añadirle una clasificación de estado nos permitirá evaluar la capacidad de reconocer cierto número de actividades sencillas con el fin de comprimir más aun los datos de envió por parte del sensor inteligente.

En la Figura 3 se plantea un diagrama de bloques para el modelo en cuestión.

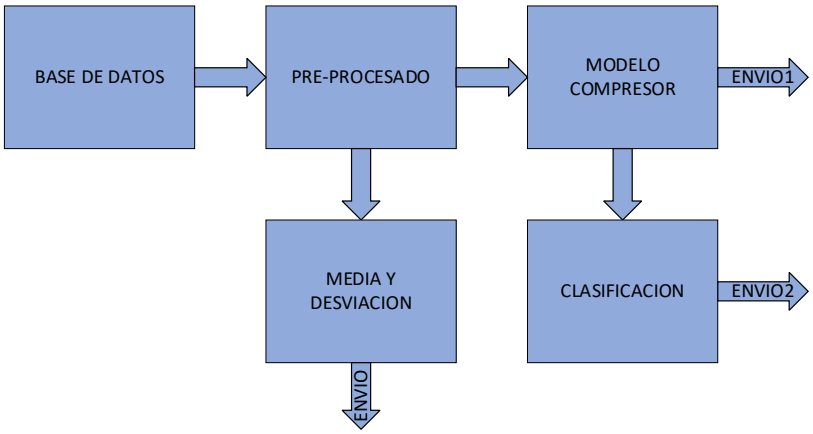


Figura 3. Diagrama bloques caso de estudio

Capítulo 3 Desarrollo Software

En este capítulo se van a desarrollar en tres apartados el tratamiento de los datos, el modelo compresor VQ y la inferencia de estado, todo ello realizado en lenguaje **Python** acompañado de los diagramas de flujo y justificaciones.

3.1. Pre-Procesado

3.1.1. Base de datos

Para el entrenamiento del modelo compresor se ha escogido la base de datos **Heterogeneity Activity Recognition Data Set [22],** al ser la que menos problemas presentaba entre sus actividades y usuarios. Todos los sensores móviles se ubican en la misma posición a excepción de los dos *smartwatchs,* los cuales no se han tenido en cuenta en la elección del sensor de estudio.

Los sensores que podemos encontrar en la base de datos son los provenientes de los dispositivos y dan su valor en m/s2 con un máximo de 40 m/s2:

- Lg Nexus4 con nexus4_1 y Nexus 4_2
- s3 con s3_1 y s3_2
- s3mini con s3mini_1 y s3_mini2
- samsungold con samsungold_1 y samsungold_2
- Smartwatch Samsung Galaxy Gear
- Smartwatch LG watches

Para nuestro caso vamos a analizar los dispositivos móviles, ya que van a ser menos sensibles al ruido en los movimientos, los relojes por estar en una zona de movimientos de alta frecuencia no van a entrar caso de estudio que se plantea para una primera evaluación del sistema.

Lo primero que haremos es identificar las frecuencias de muestreo de los dispositivos móviles, se muestran en la Tabla 5 .

Dispositivo	Max_Frecmuestreo (Hz)
Lg Nexus 4	200
Samsung S3	100
Samsung S3 mini	100
Samsung S+	50

Tabla 5. Frecuencias de muestreo sensores

Fijándonos en los tiempos de recepción de datos que han realizado los creadores de la base, podemos observar en la Figura 4 la llegada del dato y el tiempo de creación de este en el dispositivo. Han usado para representarlos un diagrama de caja, cuya mediana debe coincidir con la frecuencia de muestreo que se indica en la Tabla 5.

Los dispositivos que mejor cumplen los plazos son los Samsung S3 y Samsung S3 mini, siendo los que pasan al siguiente punto de análisis. El siguiente paso es ver la variación de los sesgos al evaluarlo en distintas posiciones como aparece en la Figura 5.

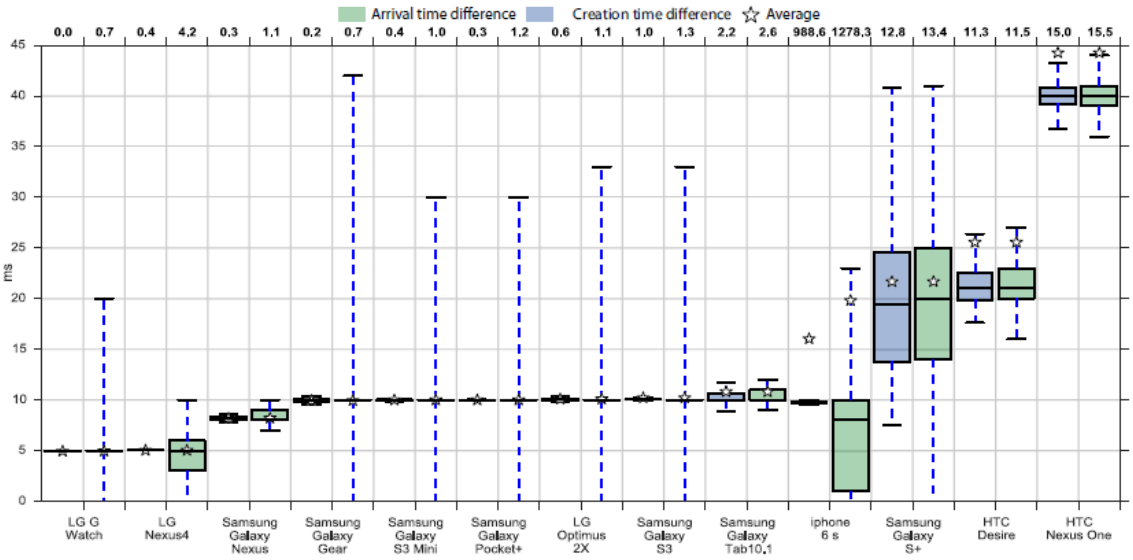


Figura 4. Intervalos de tiempo entre muestras consecutivas estimadas utilizando el tiempo de creación (azul) y la llegada tiempo (verde) [22]

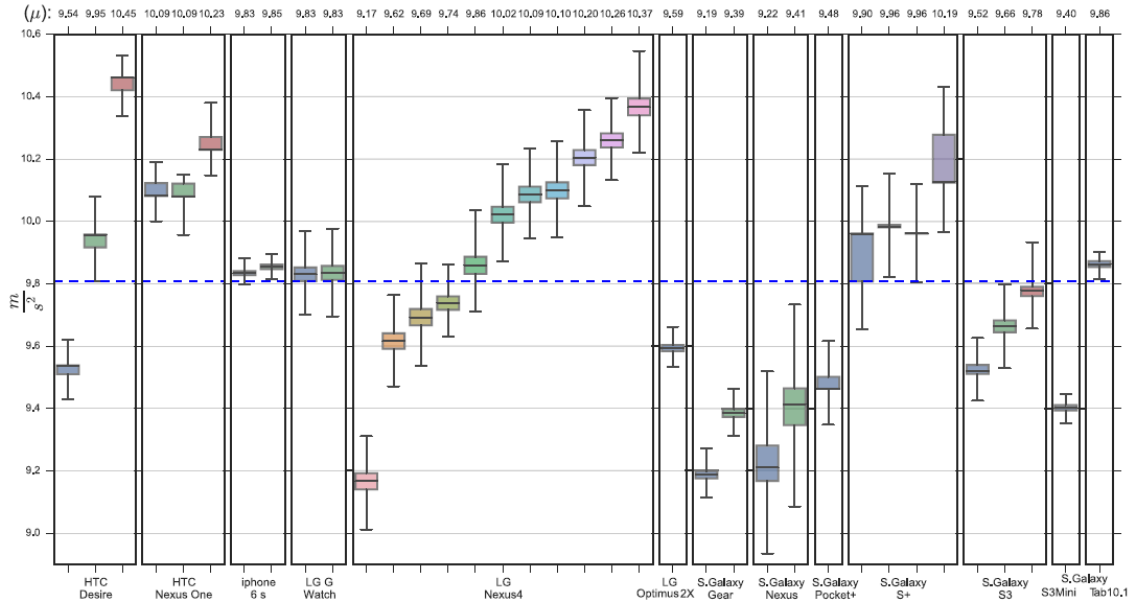


Figura 5. Sesgos de acelerómetro de dispositivos móviles en seis orientaciones diferentes en condiciones estáticas [22]

Los sesgos del S3 en los que han sido evaluados tres dispositivos distintos nos han proporcionado unos valores más cercanos al valor referencia de aceleración g[1],. Mientras que para el S3_mini solo se disponía de uno.

Con el sensor escogido Samsung S3 tanto el s3_1 como el s3_2 se va a llevar a cabo el proceso de entrenamiento del modelo compresor VQ explicado a continuación.

[1] g índice aceleración: 9.81 m/s^2

3.1.2. Abstracción

Para evaluar esta base de datos se va a realizar un pre-procesado de los datos en crudo, este va a ser llevado a cabo siguiendo el diagrama de flujo de la Figura 6.

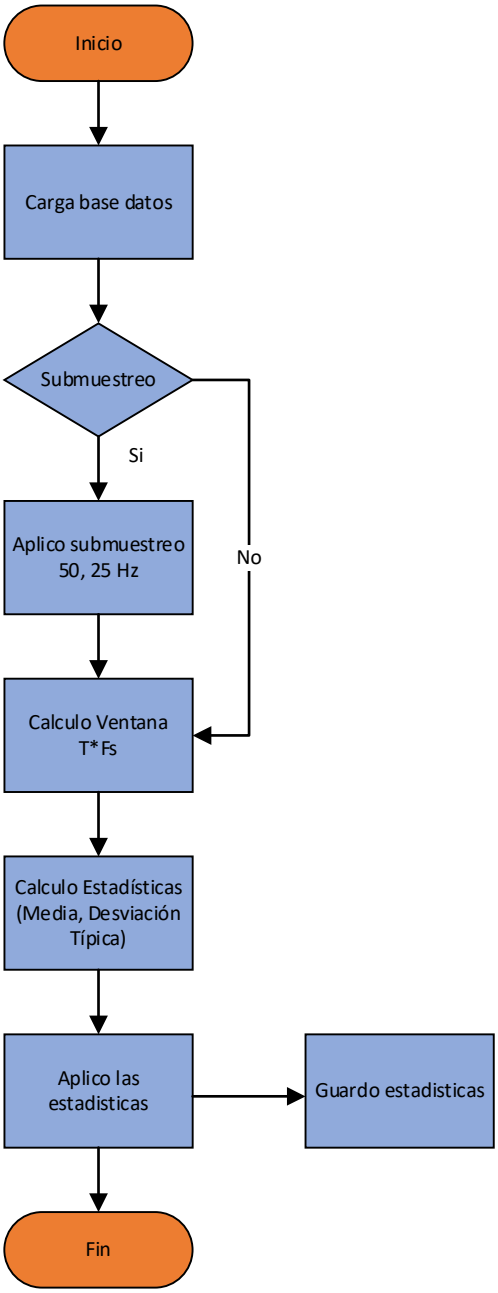


Figura 6. Diagrama Flujo Pre-Procesado

Organizaremos los datos estructurados en ventanas, en un primer caso sin deslizamiento, de manera que entren a la red concatenadas con el movimiento anterior. Para aumentar la cantidad de datos debido a la alta frecuencia de muestreo se evaluará disminuir la frecuencia con un factor 2 y 4 veces más pequeña que la del sensor. También se plantea la variación temporal del tiempo de muestra para los valores 0.6, 1, 1.6 segundos. Se escogen estas para ver el comportamiento de reducir o aumentar los datos de una ventana, el hecho de coger 0.6 ó 1.6 es para que salga un número entero de muestras en la ventana para todos los rangos de submuestreo.

Además de disponer los datos en forma de ventana se calculan la media y desviación típica para cada una de ellas, con el fin de restarle el primer parámetro y dividirlo por el segundo aplicando un cierto nivel de abstracción al dato.

El dato de media y desviación típica se enviará junto al resultado de la compresión para que el usuario final disponga de la codificación del dato y pueda reconstruir la original.

Estos datos estadísticos estarán codificados como un float de 16 bits, el menor número de bits que permite este tipo de estructura, así pues, podremos operar con mayor precisión que usando enteros.

Otro punto de estudio es la cantidad de componentes de eje del sensor que introduciremos como dato de entrada.

- Composición Full: Componente x, y, z conjuntamente.
- Composición Single: Cada una de las componentes de manera individual.

La diferencia principal está en el número de datos que compondrá el dato de entrada y la cantidad de datos estadísticos a enviar.

$$Windowsize_{Full} = 3(T \times fs) + Media + Std$$
$$Windowsize_{single} = 3(T \times fs + Media + Std)$$

Como podemos apreciar implica mandar dos medias y dos desviaciones típicas más por el canal de datos y tener un modelo de reducción de dimensión para cada componente.

Debido al alto número de dimensiones de los datos ya que el tamaño de ventana puede ir desde 15 a 160 se aplica un PCA de dos componentes para representar los datos en un espacio plano 2D, además se utilizará luego para comparar contra el modelo VQ.

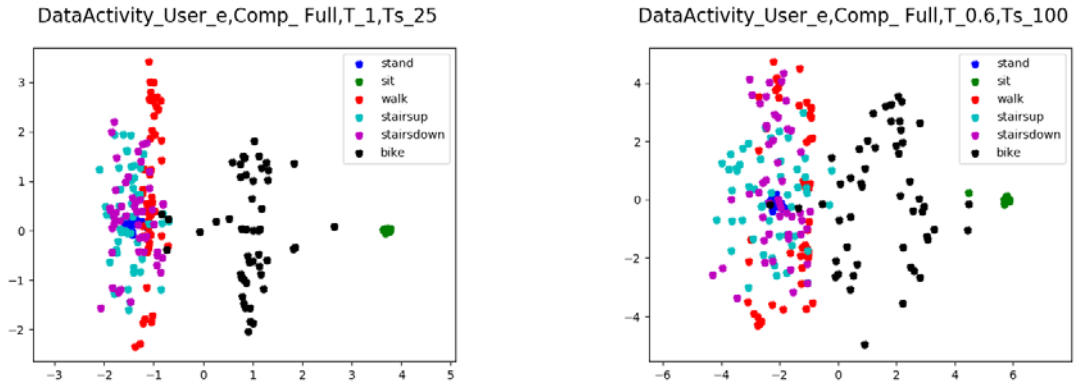


Figura 7. Representación Ventanas, usuario "e" para distintas frecuencias de muestreo (Hz) y tamaño de muestra (s)

En la Figura 7 podemos ver un ejemplo de representación de muestras de ventanas de uno de los usuarios en su representación Full y con las actividades que realiza, en los Anexos Análisis base de datos se adjuntan las gráficas de cada uno de los usuarios para T=0.6 s y Fs=25Hz, a modo de comparativa entre ellos.

3.2. Modelo Vector Quantization

3.2.1. Modelo clásico VQ

Este tipo de compresión es una técnica de comprensión con pérdida que utiliza un algoritmo de decisión competitivo, se seleccionará un valor de una lista finita (*codebook*), tabla de centroides, para representar un dato de entrada.

Cada punto de entrada tiene una representación en un espacio de n-dimensiones que conformara el conjunto de los centroides, este espacio será definido por el algoritmo de manera que se trocee optimizando las regiones para que los datos queden dentro de las regiones [28] (regiones de Voronoi).

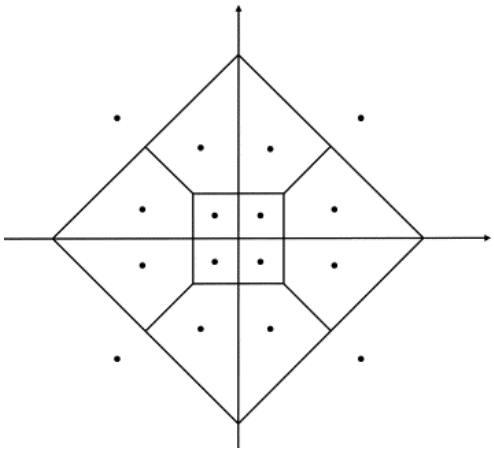


Figura 8. Regiones de Voronoi con Vector quantization [28]

El problema de esta técnica es que fracciona el mapa de los centroides de forma similar a como funciona un conversor AD[2], desaprovechándose parte del mapa si los datos solo se concentran en una zona. Este problema es el que se pretende mejorar al añadir una segunda competición basada en una magnitud que haga que los centroides se vayan moviendo.

3.2.2. Modelo adaptativo VQ

Añadiendo la capacidad al sistema compresor clásico [28] de poder evaluar una serie de magnitudes para recalcular la posición de los centroides, conseguiremos ajustar estos para conseguir una codificación superior a usar el método convencional.

Esta magnitud puede venir definida por medio de:

- Valor de entrada.
- Resto de valores de entrada.
- Cálculo de función objetivo.

Para los primeros casos, se define un mapa de magnitud que contendrá los valores correspondientes al dato de entrada, mientras que el tercer caso necesita definir una función a nivel interno que procese los datos locales y proporcione una magnitud calculada a partir de los datos dentro de la región de Voronoi.

[2] AD: Analógico Digital

Esta nueva implementación nos permite realizar una segunda competición a nivel local con las dos mejores [18] neuronas que han ganado la competición clásica por distancia euclídea.

En este caso vamos a emplear la solución de función objetivo ya que no se dispone de ningún mapa de magnitudes creado, vamos a evaluar tres funciones distintas:

- Unidad: componiendo con unos el vector magnitud.
- ESCL (Error Sensitive Competitive Learning): componiendo el vector con el Qerrormedio del centroide.
- FSCL (Frequency Sensitive Competitive Learning): componiendo el vector con la frecuencia de activación del centroide.

El primero funciona como un método K-means al realizar una competición basada en la BMU, mientras que ESCL distribuye los centroides de manera que se obtiene un error de cuantificación medio en cada región lo más parecido posible en toda la distribución de los datos. FSCL por otra parte busca igualar las activaciones de los centroides con los datos en sus regiones de Voronoi.

3.3. Algoritmo VQ adaptativo

Se va a construir un algoritmo en **Python** que forme el modelo adaptativo VQ siguiendo el diagrama de flujo de la Figura 10 , Figura 11 y concluyendo en el fichero pymscl.py adjunto en los Anexos Algoritmo MSCL.

Una vez que los datos han sido preprocesados son introducidos al entrenamiento cuyo primer paso es crear una tabla como la de la Figura 9. Esta puede ser completada en primera manera aleatoria con datos de la matriz de entrada o calculando la media de 1000 muestras y añadiéndoles un offset de ± 1 para forzar a todas las neuronas a moverse. A su vez inicializamos el vector de magnitud a uno de tal manera que el primer ciclo se comportara como un K-means.

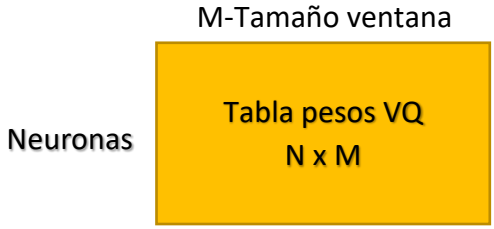


Figura 9. Composición tabla pesos VQ

Para el siguiente paso la competición global se calcula el qerror (Ecuación 1) de todas las neuronas con cada dato de entrada, consiguiendo así una lista con los qerror de cada neurona para poder buscar la mejor unidad de correspondencia o BMU (best match unit) Ecuación 2. Hasta aquí seria el uso normal de VQ y es por eso por lo que nosotros también nos quedaremos con la siguiente mejor, la NBU (next best unit), para realizar una competición local entre ambas.

$$Qerror = \sum_{i=1}^{n} (windowdata - centroid)^2$$

Ecuación 1

$$BMU = Argmin(Querror\ (w_i))$$

Ecuación 2

El resultado de la competición local determinara que neurona es la indicada para añadir su valor de qerror a la matriz temporal de acumulación que al acabar el ciclo de entrenamiento será la que cambie los valores de los pesos de la neurona. El cálculo LBMU Ecuación 3 (local best match unit) se lleva a cabo multiplicando el valor qerror de las neuronas ganadoras por la magnitud de estas, ubicadas en el vector de magnitud.

$$LBMU = argmin(Vmag(k,t) \times Querror_k^2(t))$$

Ecuación 3

Una vez conseguida la LBMU tendremos la neurona ganadora de la competición local, evitando pues que sólo las neuronas con menos distancia euclídea se lleven todas las activaciones.

Recorremos todos los datos de entrada para calcular todas las LBMU y se actualizan las veces que ha ganado cada una, así como acumular la diferencia entre la señal real y el centroide de la LBMU en la matriz de datos acumulados Ecuación 4.

$$MATDW = MATDW_k + (x(t) - w_k(t))$$

Ecuación 4

Al final del ciclo se calcula el factor de aprendizaje Alpha Ecuación 5 para actualizar los pesos, de manera que decaiga a cada ciclo de entrenamiento:

$$\propto = \propto_0 + (\propto_F - \propto_0) \times \left(\frac{t}{T}\right)^{1/2}$$

Ecuación 5

Los pesos se actualizan Ecuación 6 para la siguiente época (ciclo).

$$weight_j(t+1) = w_j(t) + \propto (t) \times \frac{MATDW}{Index\_win}$$

Ecuación 6

Para el cálculo de la magnitud volvemos a pasar todos los datos de entrenamiento, pero esta vez solo calculando la BMU, acumulando el qerror y las veces que se activa la neurona. Con estos dos valores por centroide calculamos la función magnitud correspondiente, en este caso se han implementado la unidad, FSCL y ESCL.

Con unidad el método funcionaría como un k-means actualizando continuamente su función de magnitud a uno para que siempre gane la BMU la competición local.
Mientras en la función FSCL la frecuencia de activación es la magnitud, promoviendo un nivel de activación similar en todas las neuronas. Por último, el ESCL (de las tres la más interesante para los temas de compresión) calcula la magnitud dividiendo el qerror acumulado por cada centroide con la frecuencia de activación para sacar el qerror medio de su región de Voronoi.

Este proceso se lleva a cabo con los datos de validación, esto nos permite extraer información del qerror medio cometido o la frecuencia de activación de los datos, de forma que veamos la evolución del sistema.

La condición de finalización del entrenamiento puede venir definida por el número de ciclos que hayamos fijado o por medio de una desconexión automática cuando el qerror no cambia de un ciclo a otro en un cierto porcentaje.

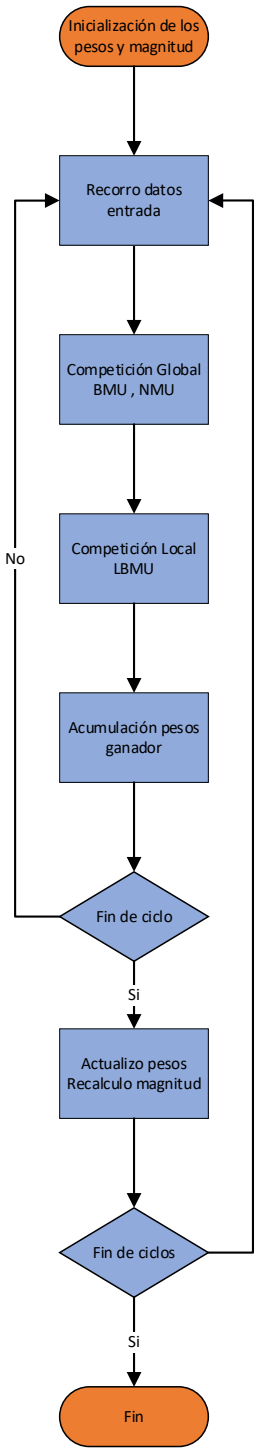


Figura 10. Diagrama Flujo entrenamiento VQ adaptativo

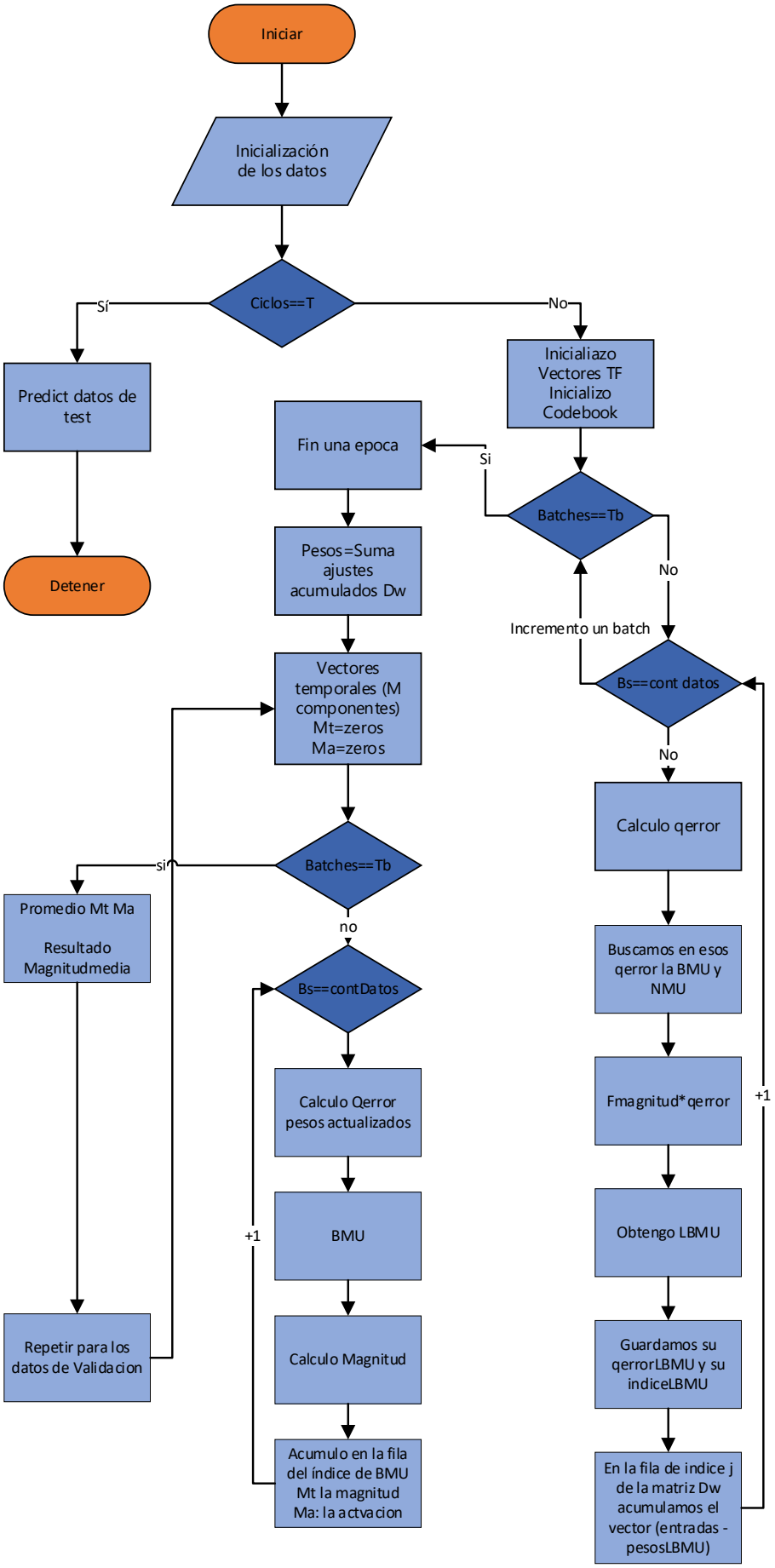


Figura 11 Diagrama Flujo entrenamiento VQ extendido

3.4. Parámetros de estudio modelo

Con el fin de facilitar la prueba de los distintos parámetros del pre-procesado y del modelo VQ se ha estructurado el código para recibir los parámetros de un fichero Json para poder simular distintos casos de manera automática y el volcado de sus resultados en gráficas o ficheros csv.

Los parámetros que van a evaluar en las simulaciones del modelo se detallan en la Tabla 6 y su evaluación se encuentra en el apartado 4.1.

Parámetro	Valores
Tiempo de ventana (T)	[0.6,1,1.6] [s]
Frecuencia de muestreo (Fs)	[25,50,100] [Hz]
Tipo de composición ventana	Full (x-y-z) o single (x,y,z)
Numero de neuronas	[1024,2048,4096,8192]
Función objetivo (TF)	Unity, FSCL, ESCL
Alpha ini	0.01
Alpha end	0.00001
Tepoch	50
Trainsize	65%
Valsize	20%
Testsize	15%

Tabla 6. Parámetros Entrenamiento

Se han dejado fijos ciertos parámetros como los porcentajes de datos de la base para realizar las tareas de entrenamiento, validación y test, así como el factor de aprendizaje para que sea pequeño y no cause una organización abrupta en los primeros ciclos de entrenamiento.

Al final del entrenamiento vamos a tener una tabla de centroides de tamaño N_neuronas x Tamaño de ventana, con una resolución de dato float16 dando lugar a un tamaño en bits Ecuación 7:

$$Vqtable(bits) = 16 \times (N_{neuronas} \times TamañoVentana)$$

Ecuación 7

Esta tabla tendrá que estar alojada dentro del sensor inteligente, y fuera para poder reconstruir los datos mediante el envío del código de la neurona ganadora. Este código de envío será un número entero con el índice de la neurona ganadora en la evaluación, el tamaño de este viene descrito por la Ecuación 8:

$$Bits_{enviocentroide} = \frac{\log(N_{neuronas})}{\log(2)}$$

Ecuación 8

3.5. Clasificador de estado

Si bien pues ya hemos conseguido reducir el tamaño de los datos de envió en función de la codificación de los parámetros estadísticos y un código de nbits para la neurona.

Aun podemos buscar reducir al mínimo la cantidad de datos de envío si el caso de uso no requiere extraer los datos del sensor, sino más bien realizar una monitorización

del estado, podemos plantear una segunda abstracción con la actividad registrada previamente.

Para registrar las actividades será necesario entrenar una tabla de consulta siguiendo un proceso como el que se refleja en el diagrama de la Figura 12 .
Esta tabla luego será transmitida al receptor para poder decodificar la señal comprimida por el VQ al tener el código de la neurona ganadora y la tabla previamente entrenada con los centroides.

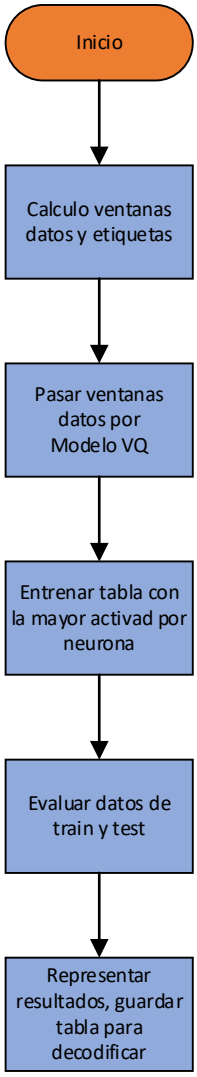


Figura 12. Diagrama de flujo Clasificación

Esto nos permite añadir otro nivel de reducción de dimensionalidad a costa de otra tabla de consulta tanto dentro del sensor como en el equipo externo, el tamaño vendrá definido por el número de neuronas que tenga nuestro modelo VQ.
Cuanto más pequeño sea éste, más centroides compartirán las mismas actividades y dará lugar a más errores de clasificación.

Capítulo 4 Análisis y resultados

En este capítulo se lleva a cabo un análisis del comportamiento del modelo compresor y la clasificación para las combinaciones de entrenamiento que se han llevado a cabo.

4.1. Modelo Vq

Para el proceso de entrenamiento se ha desarrollado un script (Anexos Scriptentrenamiento automático) que se comporta como el diagrama de flujo de la Figura 25, que junto al json (apartado Anexos Fichero configuración) evalúa de manera automática los distintos casos del modelo.

A esto se le añade un guardado de los resultados como gráficas de entrenamiento o parámetros y error medio cometido para todo el conjunto de datos en un csv.

Vamos a evaluar las tres funciones objetivo para ver como impacta el parámetro más relevante que será el error cuadrático medio (MSE). Este nos ofrece la información acerca del error que se está cometiendo pasando los datos por el reductor de dimensionalidad y decodificando con las estadísticas aplicadas anteriormente.

Para ver la diferencia de este resultado se ha evaluado tanto para los propios datos de entrenamiento como para los de test.

A costa de buscar un mse menor, será necesario implementar más neuronas subiendo la cantidad de bits por envío y aumentando el tamaño del codebook, aquí se va a dar un compromiso entre cantidad de neuronas y tamaño de la ventana.

Además de los parámetros de la red y el pre-procesado no hay que dejar de lado si queremos reducir la dimensión de todas las componentes a la vez o por tema de tamaños nos podemos permitir tener un modelo por cada componente del acelerómetro.

Por último, la función de activación nos da ese añadido que busca el proyecto al no depender solamente de la asignación de centroides por distancia mínima al dato que lo contiene.

Para dar un aire comparativo se ha ejecutado un PCA de hasta 10 componentes en todos los modelos para contrastar el error cometido (Pruebas MSCL FULL y Pruebas MSCL Single) y se han realizado gráficas de caja con el PSNR Ecuación 9 como parámetro para representar la cantidad de bits necesaria para igualar mismos niveles.

$$PSNR = 10 \times \log(RangeSensor * \frac{RangeSensor}{mse})$$

Ecuación 9

En el proceso de entrenamiento de los modelos VQ se han añadido gráficas para mostrar la evolución de los parámetros: Alpha, magnitud, cantidad de neuronas sin actividad y entropía de Shanon.

Esta última ofrece una medida de la utilización de la información, para elementos de semejantes probabilidades la entropía es máxima, en el método FSCL se ve como llega a valores más altos que para los otros dos.

Por otro lado, también se está extrayendo el qerror acumulado en entrenamiento y en evaluación, así como la frecuencia media de activación a la que tienden las neuronas del mapa al ir pasando los ciclos.

A continuación, se muestra una serie de tablas en la aparecen algunos de los resultados más relevantes, el resto se ubican en los Anexos Pruebas MSCL.

Para presentar los datos de una manera organizada se separan los resultados según la función de activación y al final se redactan unas conclusiones.

4.1.1. Magnitud Unidad

Lanzamos la simulación para el usuario de estudio “e” con la opción de composición Full y obtenemos unas respuestas como las que aparecen en la tabla Tabla 7 parte del resto de simulaciones se encuentran en Pruebas MSCL .

FS	Periodo	Neuronas	SizeVQ	BitsEnvioVQ	BitsStd	MSE_mean train(m/s2)	MSE_mean test (m/s2)
25	0.6	2048	92160	11.0	32	0.74853	0.85467
25	0.6	4096	184320	12.0	32	0.51806	0.66455
25	1.6	4096	491520	12.0	32	0.42041	0.81640
50	0.6	4096	368640	12.0	32	0.26342	0.52978
50	1.6	4096	983040	12.0	32	0.33422	0.86376
100	0.6	4096	737280	12.0	32	0.31713	0.99804
100	1.6	4096	1966080	12.0	32	0.40649	1.36035

Tabla 7.Modelo Vq, Magnitud unidad, Full, user e

Se observa la tendencia a disminuir el error que se encuentra en rango de unidades de entrada m/s2 a medida que aumentamos el número de neuronas.

En las Figura 13 y Figura 14 representamos la evolución de los parámetros de entrenamiento, en la primera podemos ver como el factor de aprendizaje Alpha decrece hacia el valor Alpha fin.

La magnitud es la unidad para todo el proceso y no varía, el índice de Shanon va tendiendo a uno sin alcanzarlo debido a la inestabilidad en la frecuencia de activación de las neuronas y vemos que no tenemos ninguna neurona inactiva.

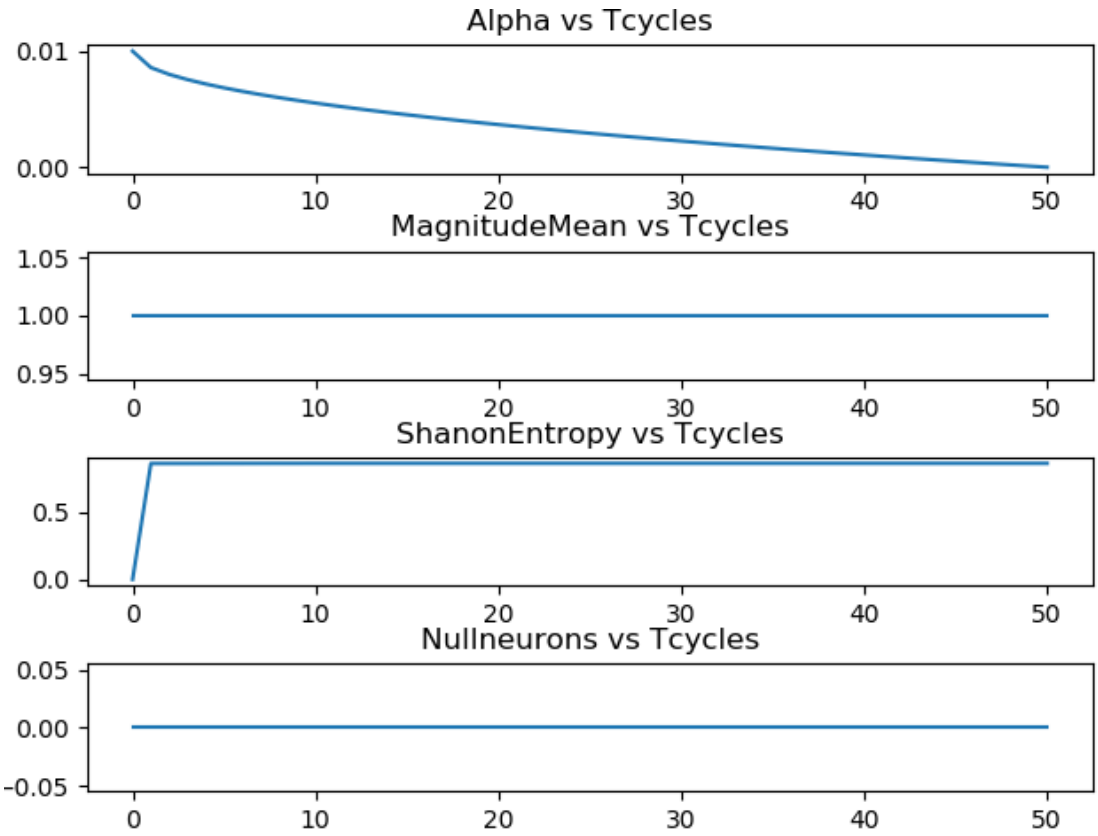


Figura 13. Evolución parámetros entrenamiento unidad, neuronas= 4096, Ts=1.6 s, fs=25Hz

Para la Figura 14 podemos ver la evolución del qerror medio acumulado para evaluación y para test, del que se percibe la caída en ambos ejes hacia la búsqueda del qerror optimo proporcionado en este caso solo por la BMU, en la frecuencia de activación se observa cómo hay neuronas que están absorbiendo muchas activaciones, no permitiendo al resto actualizar sus pesos, veremos cómo esto se corrige para el FSCL y ESCL.

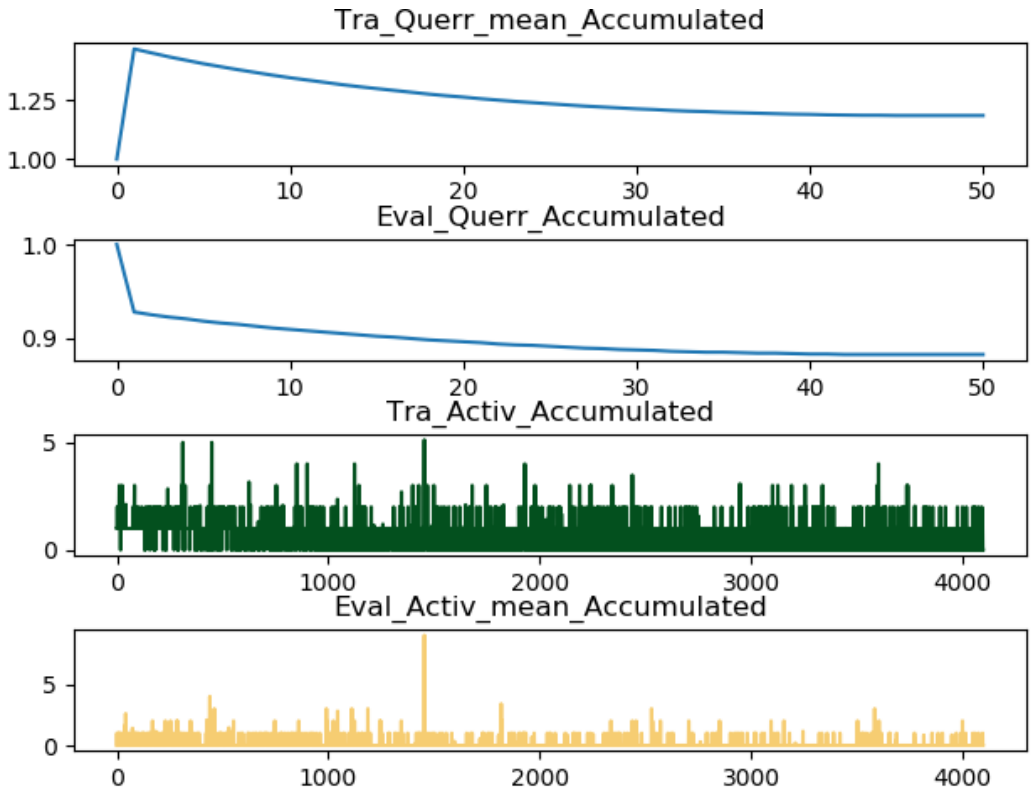


Figura 14.Qerror acumulado y Activación media de neuronas, modelo unidad, neuronas 4096, Ts =1.6 s, Fs = 25Hz

En la Figura 15 se ha representado en un diagrama de caja el PSNR calculado para el modelo tanto para PCA de una a en una sus 10 componentes siendo el valor en bit de estas representaciones [16,32,48,64,96,112,128,144,160] y del modelo unidad con tres configuraciones de neuronas incluyendo entre paréntesis el número de bits de envió necesarios, comparando con los bits necesarios para igualar el PSNR con PCA que demostrado la fuerte reducción de dimensionalidad que implica VQ.

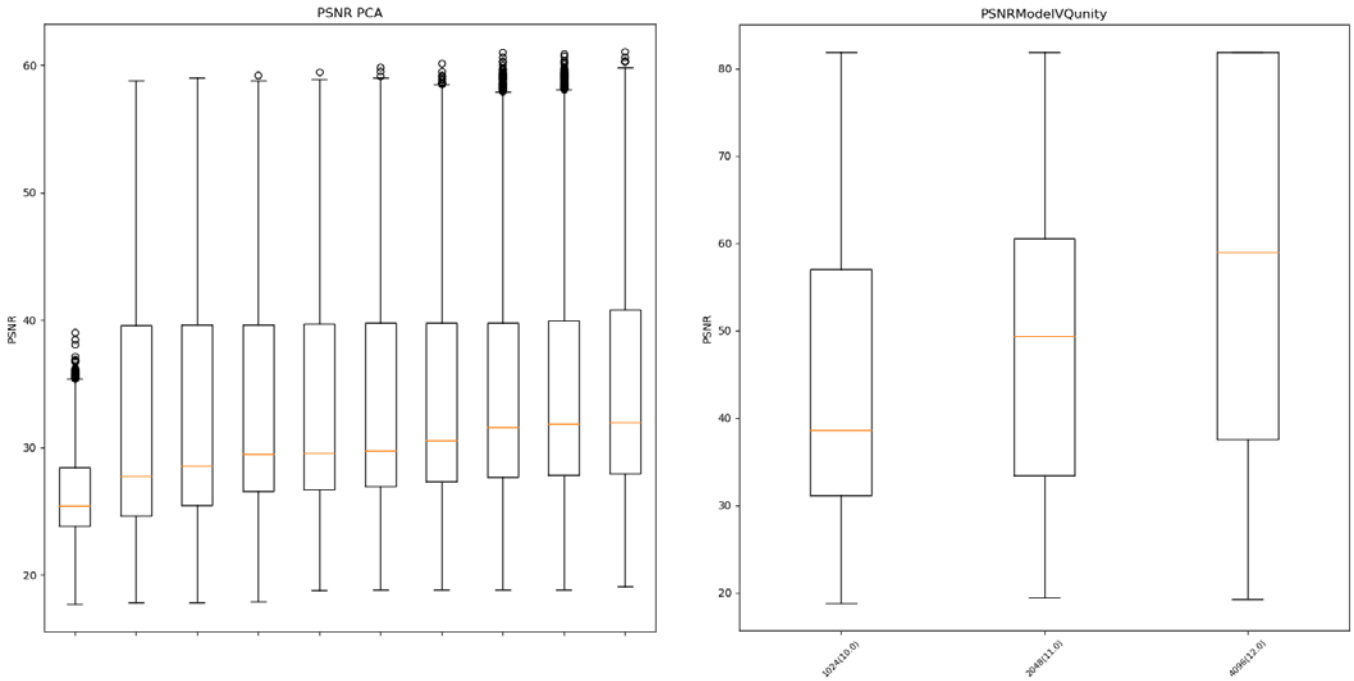


Figura 15. PSNR PCA 10 componentes y modelo unidad, Ts =1.6 s, Fs=25 Hz

4.1.2. ESCL

Evaluamos los datos con la función objetivo ESCL Tabla 8, con esta configuración deberá distribuir los centroides para que todos den un qerror similar.

fs	period	Neuronas	SizeVQ	BitsEnvioVQ	BitsStd	MSE_mean train(m/s2)	MSE_mean test(m/s2)
25	0.6	2048	92160	11.0	32	0.74462	0.85009
25	0.6	4096	184320	12.0	32	0.52880	0.66699
25	1.6	4096	491520	12.0	32	0.42895	0.80810
50	0.6	4096	368640	12.0	32	0.27514	0.49633
50	1	4096	614400	12.0	32	0.40087	0.73681
50	1.6	4096	983040	12.0	32	0.36035	0.83935
100	1	4096	1228800	12.0	32	0.40429	1.12300
100	1.6	4096	1966080	12.0	32	0.42407	1.35724

Tabla 8. Modelo Vq, Magnitud ESCL, Full, user e

En la Figura 16 ya podemos observar como la magnitud va tendiendo a cero de forma que todos los centroides estén a una misma distancia entre ellos para definir regiones de voronoi de igual qerror.

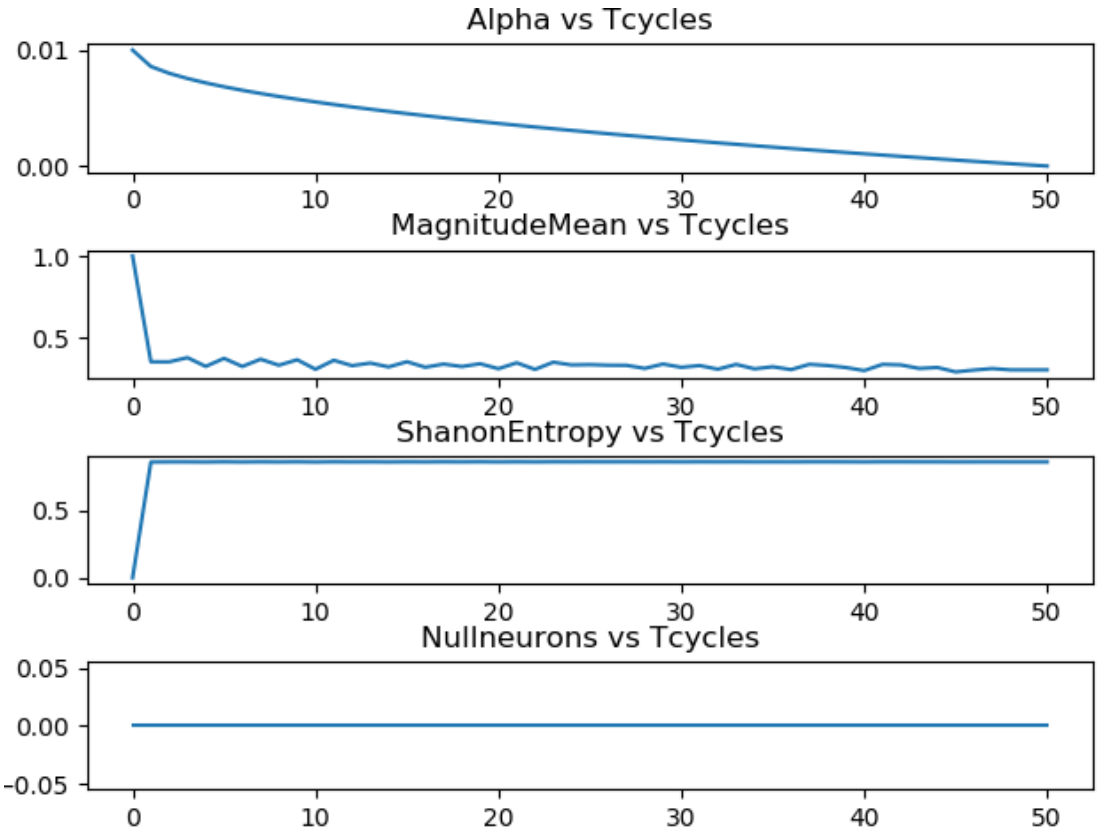


Figura 16.Evolución parámetros entrenamiento ESCL, neuronas= 4096, Ts=0.6 s, fs=50Hz

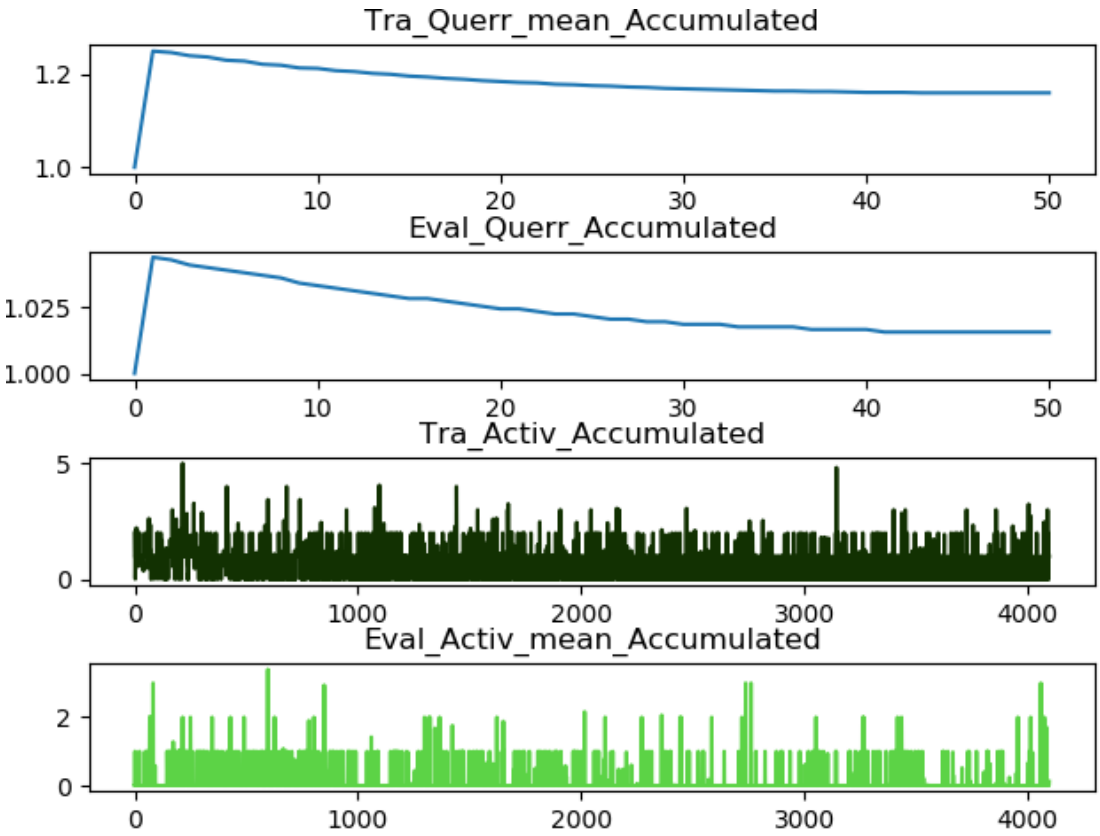


Figura 17. Qerror acumulado y Activación media de neuronas, ESCL, neuronas= 4096, Ts= 0.6 s, fs=50Hz

En la Figura 17 se observa cómo va decreciendo el qerror de los centroides a medida que pasan los ciclos de entrenamiento.
La Figura 18 hace referencia al PSNR obtenido por medio de la codificación PCA y con el ESCL, podemos observar que obtenemos una mejor respuesta ante un menor número de bits de envío.

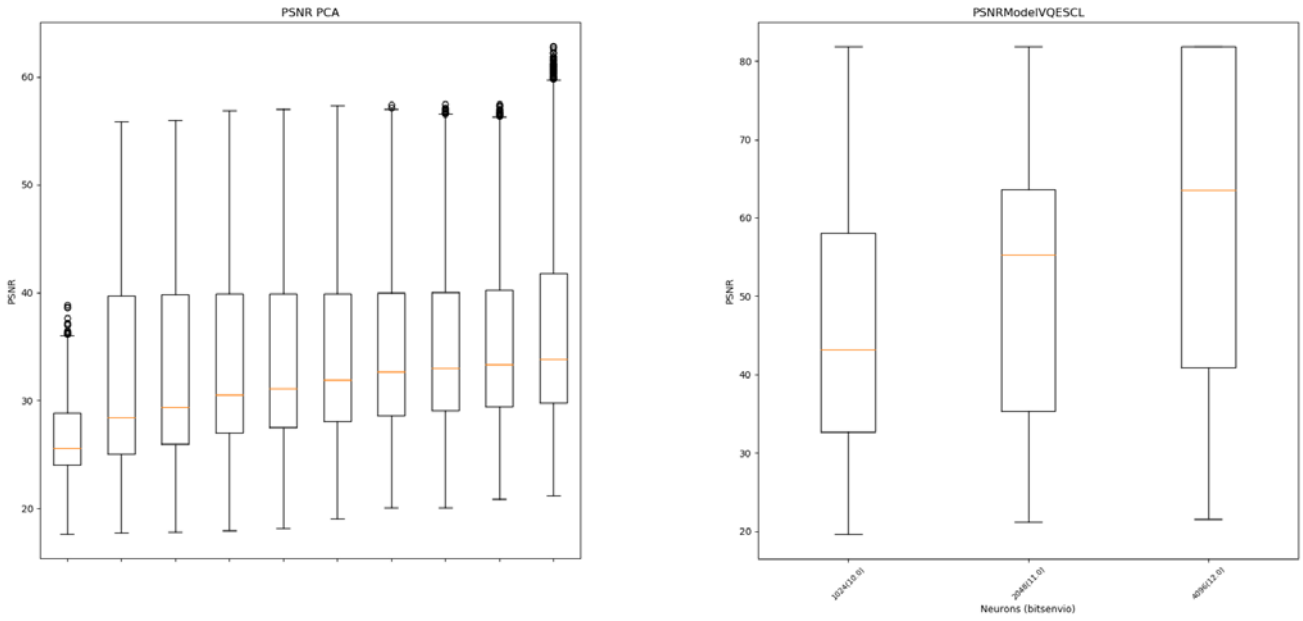


Figura 18.PSNR PCA Y PNSR ESL, Ts =0.6 s, Fs=50 Hz

4.1.3. FSCL

Por último, hemos evaluado el FSCL, del que esperamos una mejor frecuencia de activación al distribuirse los centroides buscando una activación igualitaria de todas ellas.

fs	period	Neuronas	SizeVQ	BitsEnvioVQ	BitsStd	MSE_mean train(m/s2)	MSE_mean test (m/s2)
25	0.6	1024	46080	10.0	32	0.81250	0.92089
25	0.6	2048	92160	11.0	32	0.58789	0.75048
25	0.6	4096	184320	12.0	32	0.34497	0.58251
25	1	2048	153600	11.0	32	0.54003	0.72900
25	1.6	4096	491520	12.0	32	0.41503	0.82324
50	1	4096	611440	12.0	32	0.18298	0.46435
50	1.6	4096	983040	12.0	32	0.30004	0. 74511
100	1	2048	1228800	11.0	32	0.3168	1.62304
100	1.6	4096	1966080	12.0	32	0.37744	1.37207

Tabla 9. Modelo Vq, Magnitud ESCL, Full, user e

Vemos en la Figura 19 como para esta función objetivo alcanzamos el índice más alto para la entropía de Shanon al seguir una distribución de probabilidad similar de activación para todos los centroides. Por otro lado, el valor de magnitud medio es este caso se incrementa al no tratarse ya del qerror o la unidad y ser la media de las activaciones.

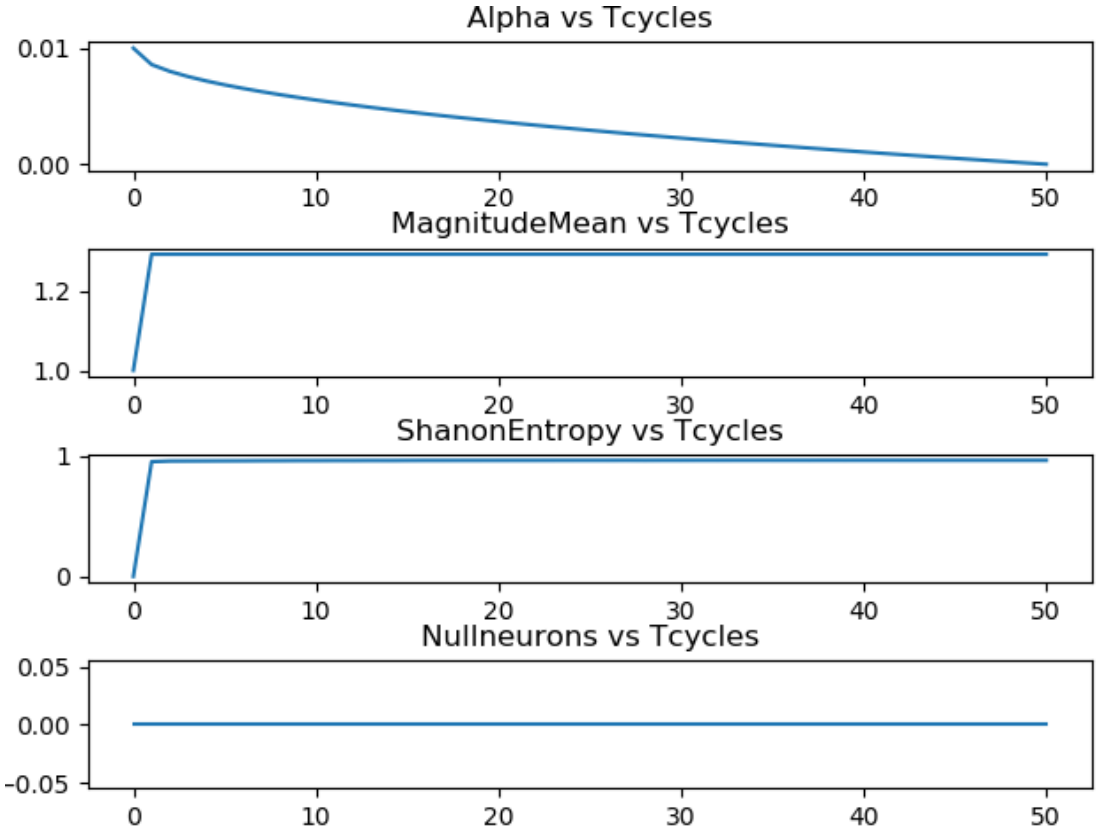


Figura 19. Evolución parámetros entrenamiento FSCL, neuronas= 4096, Ts= 0.6s, fs=25Hz

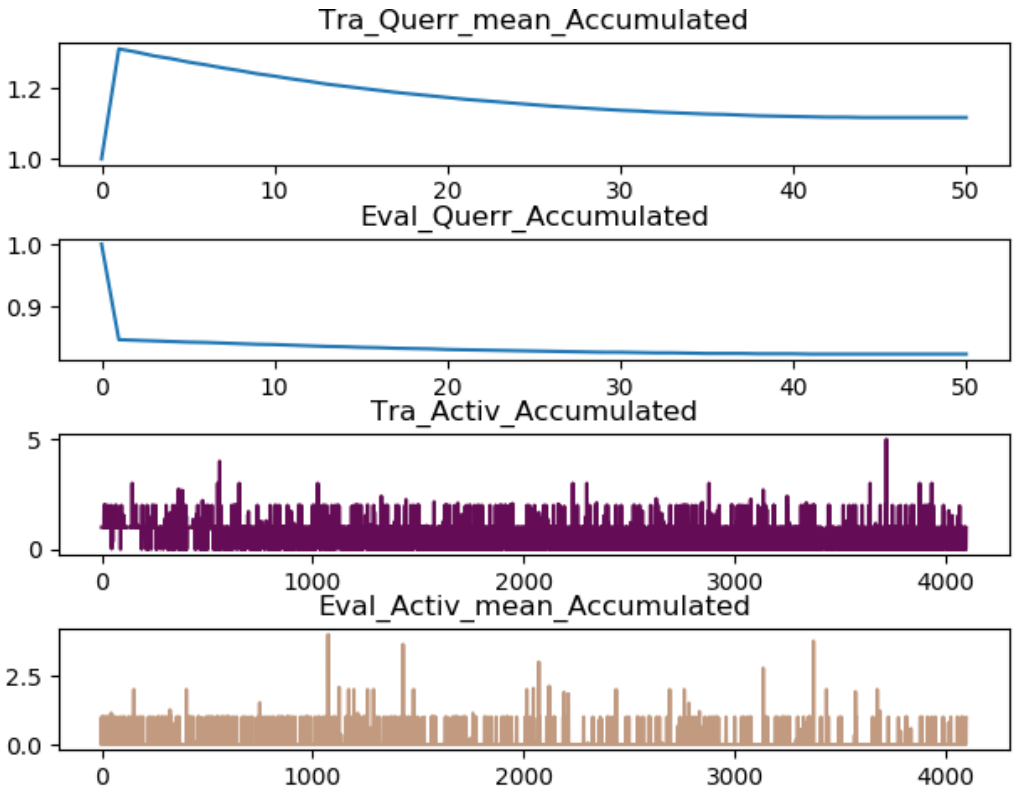


Figura 20. Qerror acumulado y Activación media de neuronas, FSCL, neuronas= 4096, Ts= 0.6s, fs=25Hz

En la Figura 20 se aprecia la activación para todas las neuronas sin producirse picos tan altos y con muchos centroides con la misma frecuencia, para esta simulación el qerror medio ha caído bruscamente tanto para el caso de validación como el de entrenamiento. A continuación, la Figura 21 con los diagramas PSNR.

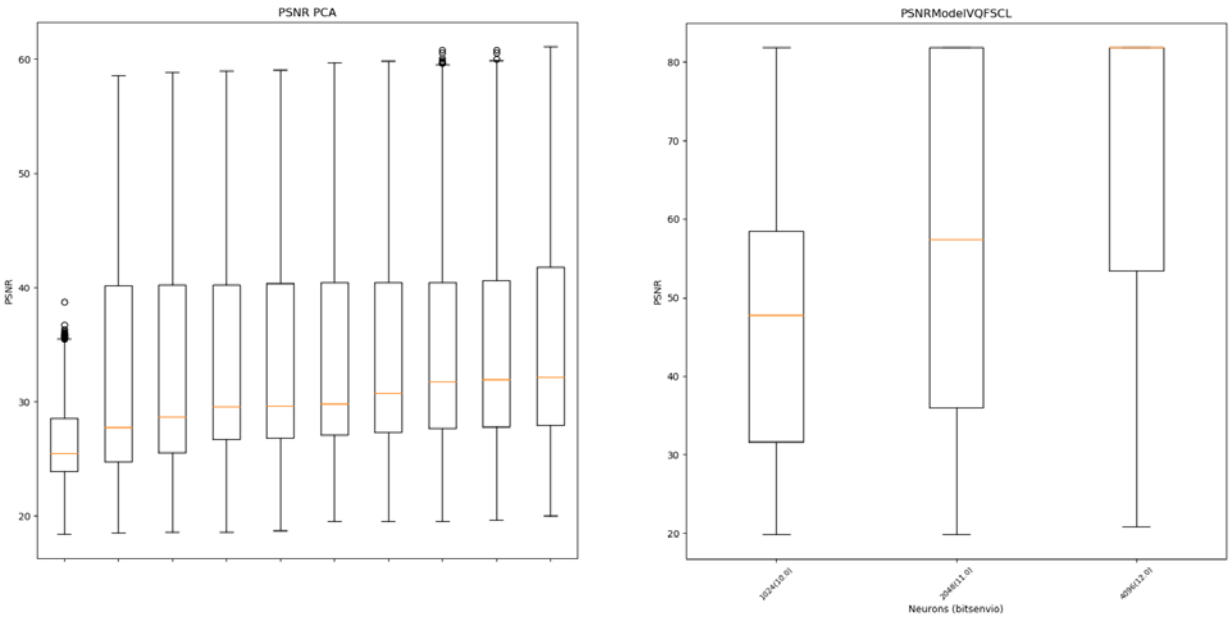


Figura 21. PSNR PCA 10 componentes y PNSR ESCL, Ts =0.6 s, Fs=25 Hz

4.1.4. Pruebas Single axis

También se han realizado pruebas para cada componente del censo incrementando el tamaño de memoria necesario para almacenar las tres tablas, estas evaluaciones se adjuntan en el Anexos al ser muy extensas.

Podríamos seguir una política de tamaño para estas tablas de tal manera que siguiera la Ecuación 10 .

$$Bits_{neuronasfull} + Bits_{stadistics} = 3 * (Bits_{neruonassingle} + Bits_{stadistics})$$

Ecuación 10

Se muestran en la Figura 22 y Figura 23Figura 20 para la componente Y del usuario e la evolución de sus parámetros durante el entrenamiento, se observa que el qerror acumulado tiende a un valor fijo siguiendo un parámetro de activación similar por parte de las neuronas para todo el ciclo, para el resto de simulaciones se trabaja con la configuración completa ya que ha dado mejor resultado en los entrenamientos previos.

Se ha simulado el comportamiento de los datos de la ventana y la distribución de los centroides para esta configuración obteniendo la Figura 24, debido a esta disposición circular, del que se intuye alguna clase de limitación a la hora de formar la base de datos, se entiende el valor de qerror más elevado que para la composición de tres componentes, a través de esto podemos pensar que las tres componentes serían los tres círculos superpuestos unos con otros con unos offset.

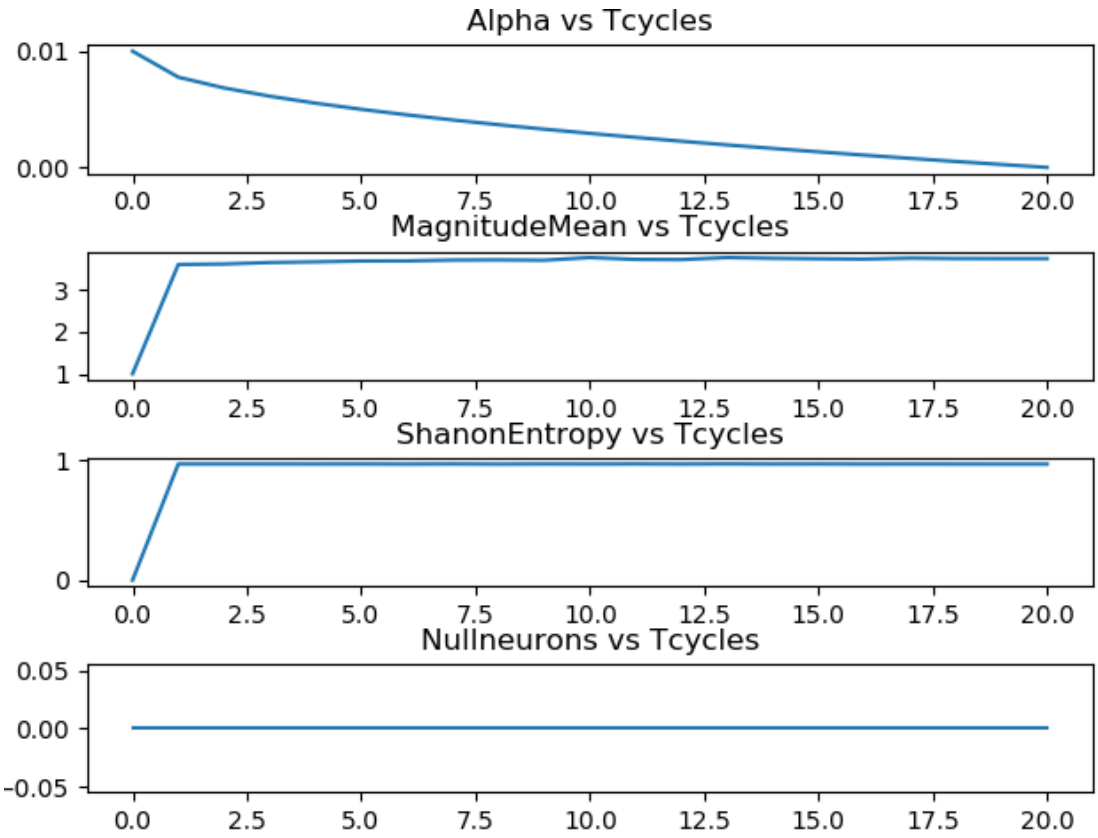


Figura 22. Evolución parámetros entrenamiento ESCL compY, neuronas=4096, T=1 s,fs= 50Hz

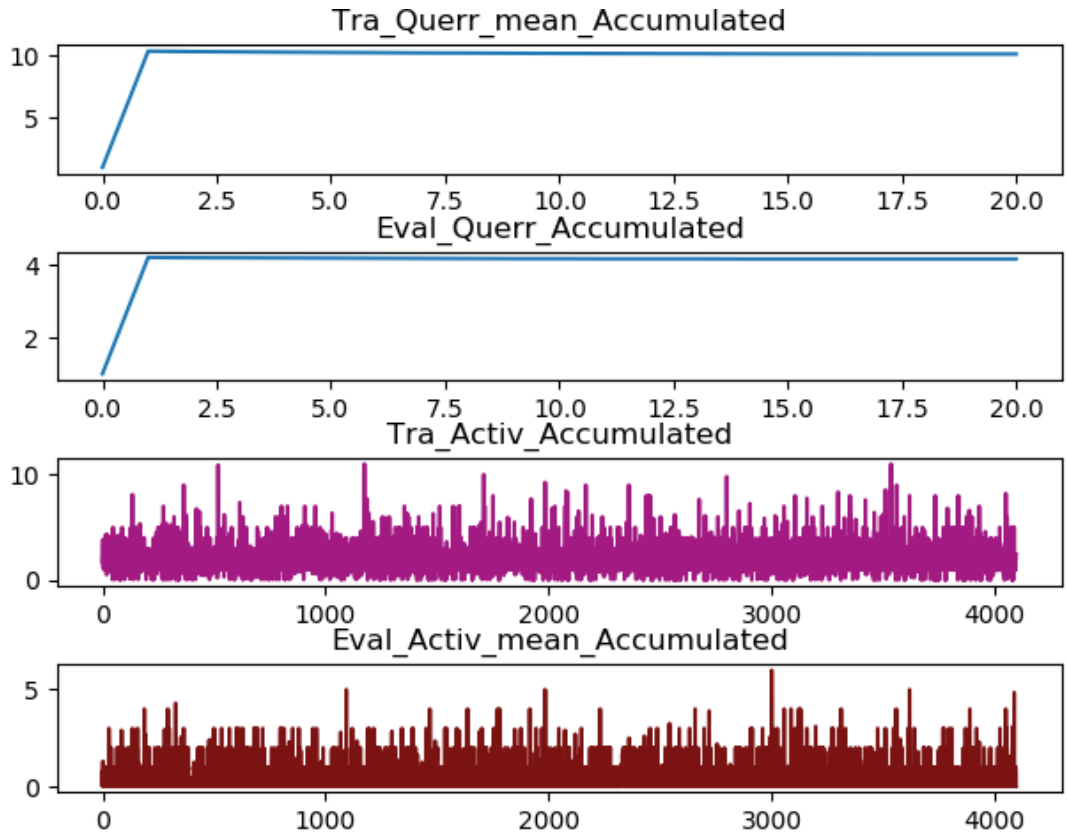


Figura 23. Qerror acumulado y Activación media de neuronas ESCL compY, neuronas=4096, T=1 s,fs= 50Hz

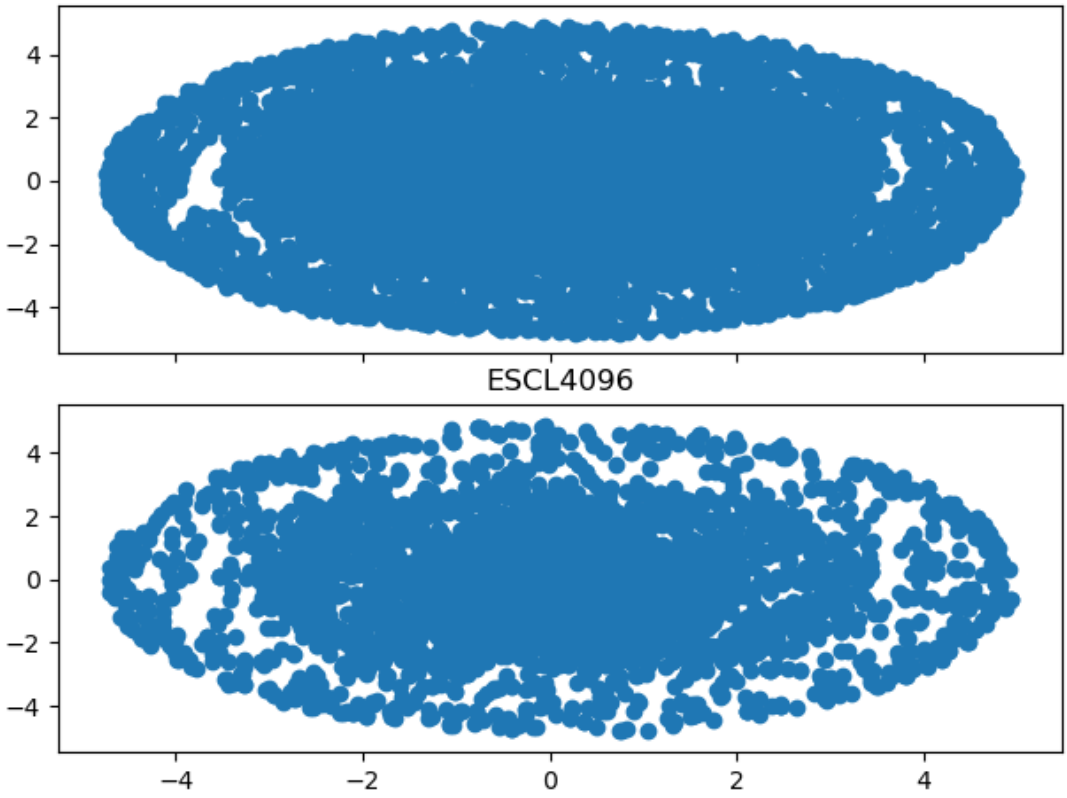


Figura 24.Datos entrada vs Centroides ESCL4096 T=1 s, fs=50 Hz

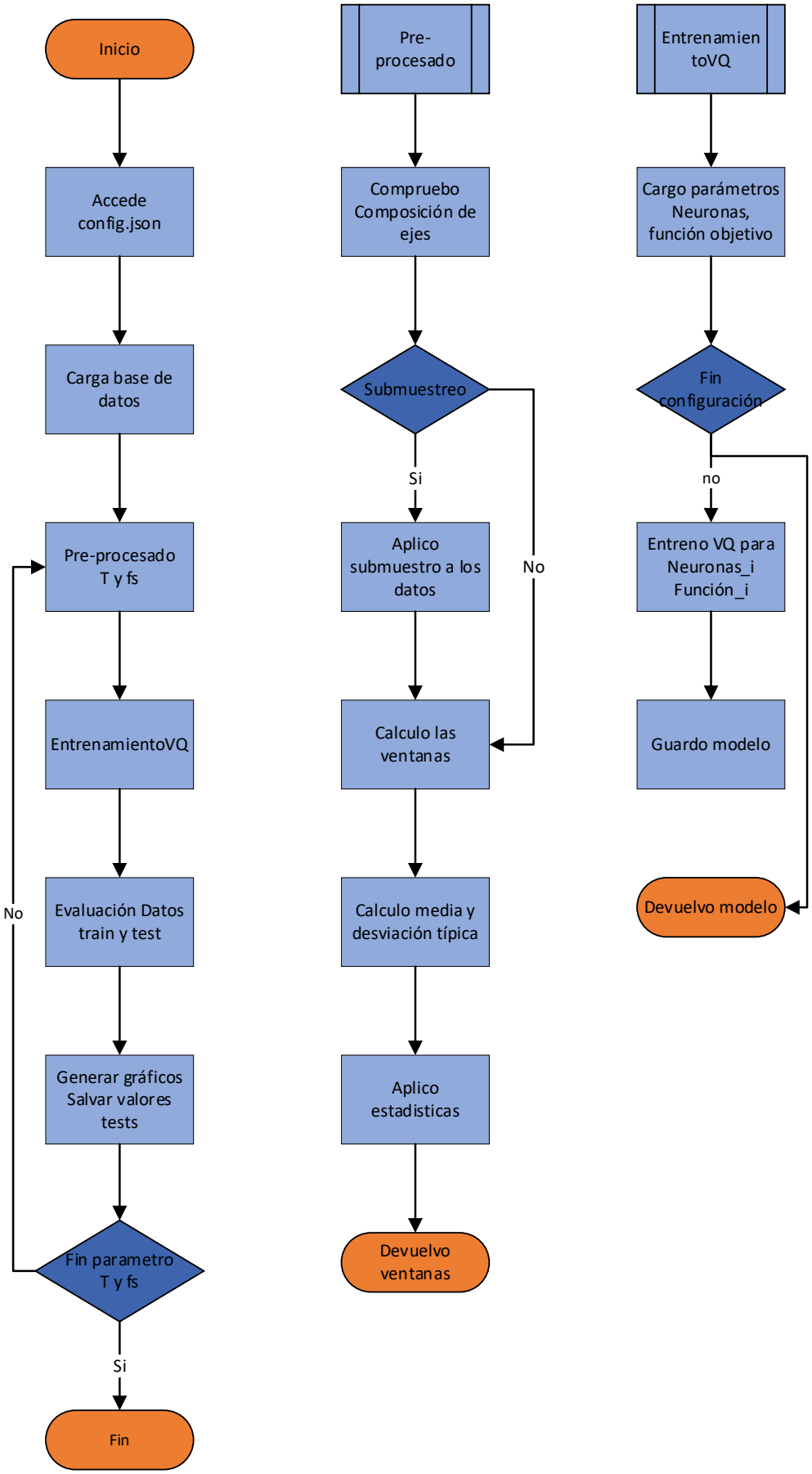


Figura 25. Sistema Auto-train

4.2. Modelo Clasificación

Con el fin de analizar el impacto de la clasificación en la reducción total del número de bits se ha planteado un modelo de clasificación por neurona más votada para asociar a cada una la actividad que más veces haya sido reconocida.

De esta manera podríamos plantearnos mandar solo la codificación de la actividad con un entero y que el decodificador tuviera la misma tabla de consulta de actividades.

Para evaluar la calidad de la clasificación se ha seguido el esquema del diagrama de flujo de la Figura 12 , llevándose a cabo una serie de simulaciones con el fin de ver si con la cantidad de datos y esas etiquetas es suficiente para una clasificación.

Obteniendo matrices de confusión para entrenamiento como las de las siguientes Figura 26.

Stand	**164**	106	101	102	111
Walk	57	**86**	59	64	65
Stairsup	36	45	**58**	35	43
Stairsdown	34	39	42	**51**	47
Bike	48	45	59	52	**82**

Figura 26. Matriz confusión train ESCL, neuronas=4096, T=1.6

Stand	**25**	24	21	15	22
Walk	16	**14**	9	6	11
Stairsup	7	9	**8**	2	12
Stairsdown	14	10	6	**5**	4
Bike	16	9	7	6	**9**

Figura 27. Matriz confusión test ESCL, neuronas=4096, T=1.6

Stand	**162**	115	110	99	98
Walk	60	**89**	55	66	61
Stairsup	39	48	**59**	31	40
Stairsdown	41	34	48	**56**	34
Bike	59	49	56	45	**77**

Figura 28.Matriz confusión train FSCL, neuronas=2048, T=1.6

Stand	**29**	16	23	20	19
Walk	9	**12**	14	10	11
Stairsup	10	6	**6**	4	12
Stairsdown	6	8	10	**7**	8
Bike	14	10	7	8	**8**

Figura 29.Matriz confusión test FSCL, neuronas=2048, T=1.6

Stand	**165**	98	91	103	127
Walk	59	**83**	53	70	66
Stairsup	43	49	**51**	40	34
Stairsdown	45	38	32	**59**	39
Bike	52	44	46	55	**8924**

Figura 30Matriz confusión train unity, neuronas=2048, T=1.6

Stand	**24**	26	14	21	22

Walk	10	**15**	7	7	17
Stairsup	8	7	**8**	10	5
Stairsdown	10	8	2	**8**	11
Bike	12	7	7	9	**12**

Figura 31. Matriz confusión test unity, neuronas=2048, T=1.6

Configuración	Precisión Train	Precisión Test
ESCL,n=4096, T=1.6, Fr=, e, s3	27.04%	21.25%
FSCL,n=2048, T=1.6, Fr=, e, s3	26.98%	26.13%
Unity, n=2048, T=1.6, Fr=, e, s3	27.41%	23.34%

Tabla 10.Resultados precisión clasificadores

En la Tabla 10 aparecen las precisiones de test y train para los datos, el resultado es bastante desalentador con los patrones que hay en la base y el método de ganadora por número de activaciones máximas.

La idea que se perseguía conseguir era definir dos casos de uso básico para este tipo de aplicación.

- Monitorización de actividad del usuario.
- Cambio de estado del usuario.

Con el caso de cambio de estado se perseguía un envió de datos mínimos notificando de manera periódica el estado, pero distando mucho del tiempo real y solo notificar de manera inmediata si se produce un cambio en el estado.

4.3. Conclusiones

El resultado de la compresión ha sido muy positivo viendo unos errores del orden de 1 m/s2 cuando el sensor cuenta con una resolución de 40 m/s2, analizando las figuras de PSNR podemos darnos cuenta del alto factor de compresión que tiene este método, ya que a nivel del PCA con una configuración de 4096 neuronas, el equivalente a 12 bits, conseguimos mejores niveles que con el PCA más bajo que con una componente de 16 bits.

Codificando en códigos del orden de 10 a 14 bits ventanas de entrada de hasta 4800 bits de entrada para el caso 100 Hz T=1,6 s, siendo una compresión del orden de cientos de bits.

Respecto a la clasificación nos encontramos cerca del valor de probabilidad aleatorio de clases aun habiendo sustituido la clase sit por stand con la esperanza de abarcar un mayor acierto. La decisión de reducir la clase sit a stand a seguido por definirse las actividades como estar sentado o estar de pie, siendo ambos casos más estáticos, con el fin de conseguir un mejor resultado de test, aunque ha sido inútil.

Se ha evaluado con un solo teléfono s3_2 y con s3 que engloba los dos terminales para el usuario e, en los Anexos Pruebas MSCL FULL se encuentran más datos de los representados en este capítulo.

En la Figura 32 y Figura 33 representamos la reconstrucción de las señales de entrada por el modelo ESCL para el usuario e con ambos terminales s3 , obteniendo estos resultados una vez aplicados a ambos los factores estadísticos que se habían usado para codificar cada ventana.

Como último intento por estudiar la versatilidad del modelo a la variación de cambios se ha predicho la señal con el modelo del usuario e para el d obteniendo los resultados que aparecen en la Figura 34 y Figura 35.
El qerror ha aumentado ya que hay zonas que no ha aprendido y cuyo centroide esta lejano a la zona de activación.
Esto corrobora la diferencia de los usuarios para realizar los mismos patrones, por ello para intentar conseguir un modelo que cubra el mayor rango posible es necesario entrenarlo con todos los casos disponibles.

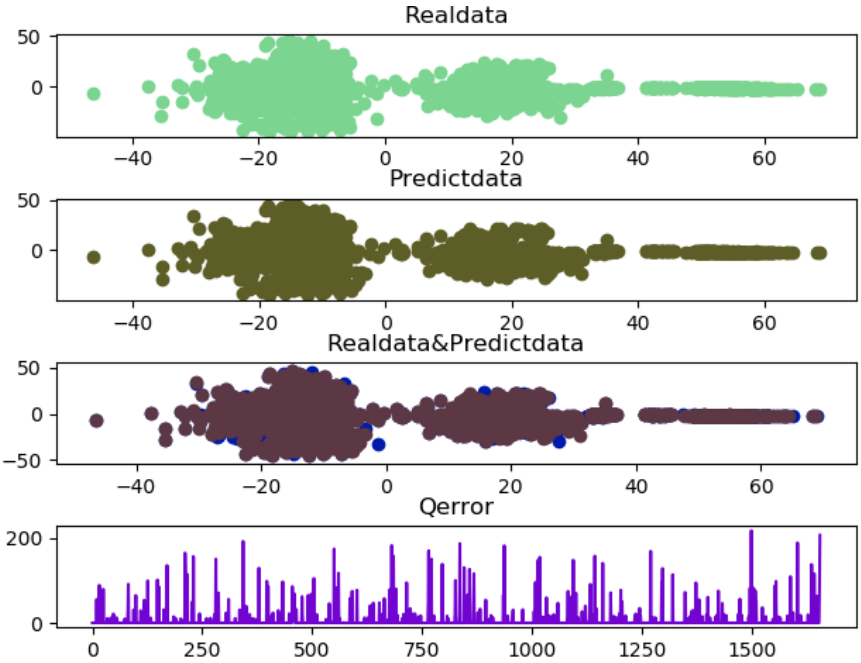


Figura 32. Predictor train ESCL 4096, T = 1.6 s ,Fs = 100 Hz

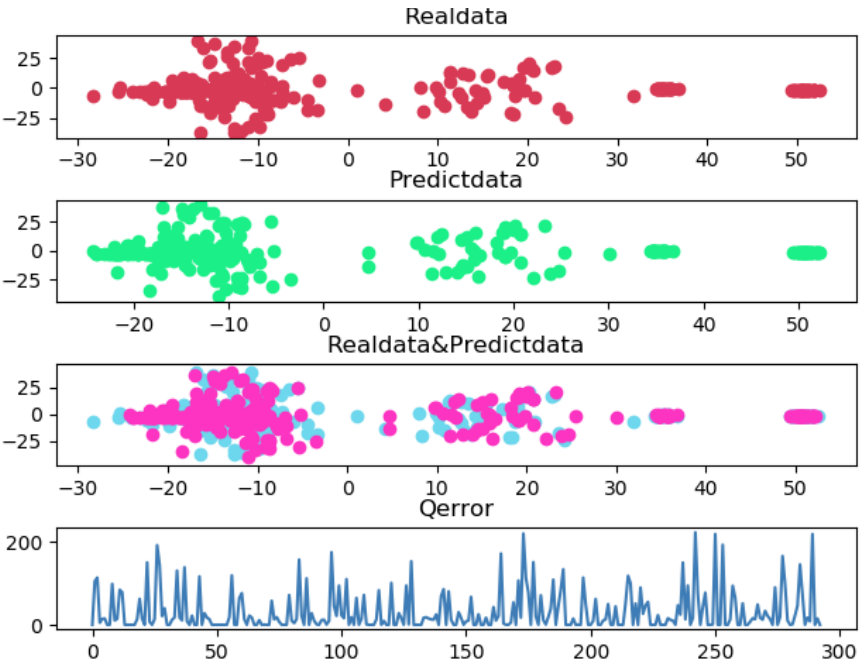


Figura 33. Predictor test ESCL 4096, T = 1.6 s ,Fs = 100 Hz

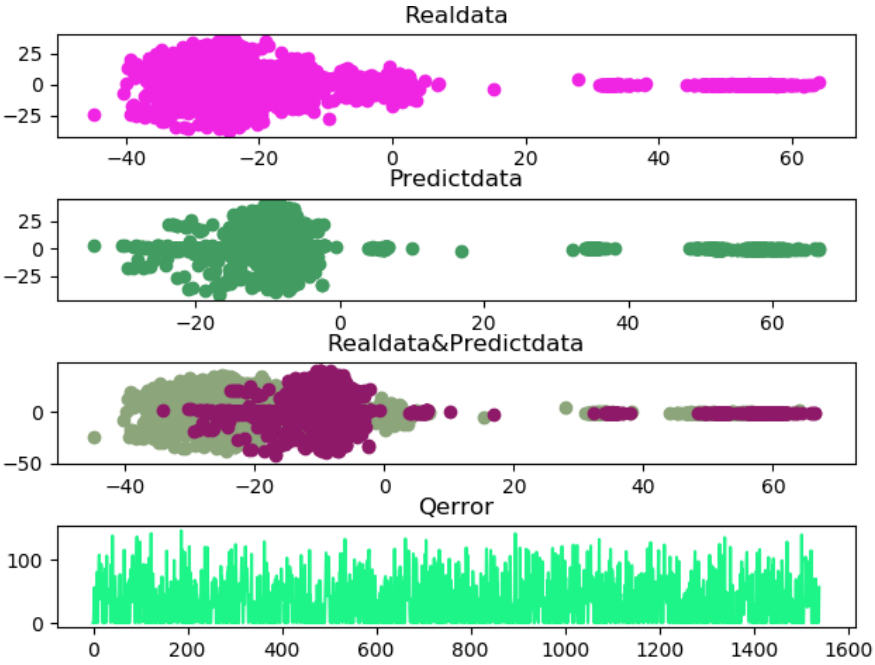


Figura 34. Predictor train user d ESCL 4096, T = 1.6 s ,Fs = 100 Hz

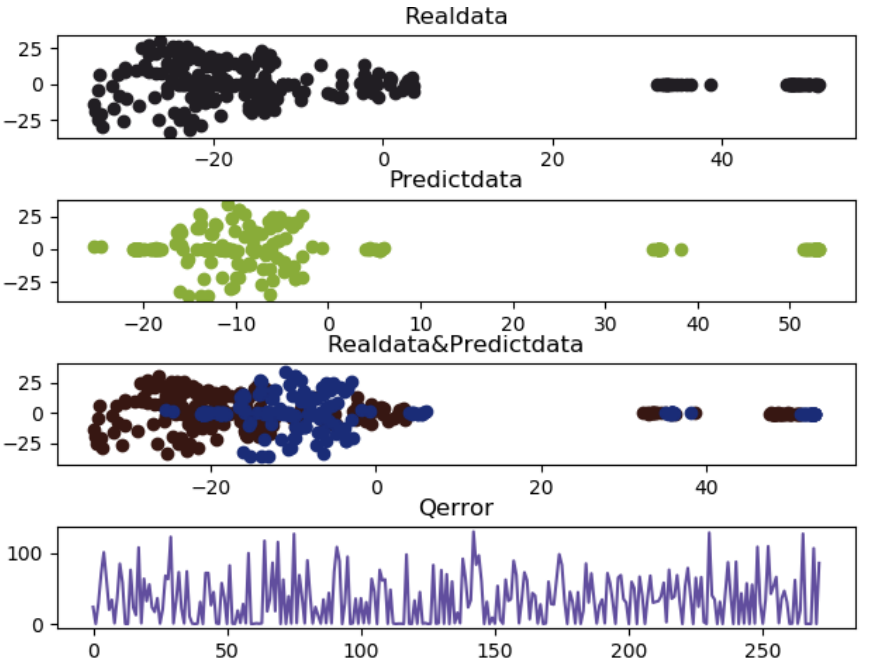


Figura 35. Predictor test user d ESCL 4096, T = 1.6 s ,Fs = 100 Hz

Capítulo 5 Implementación hardware

El último de los objetivos planteados es introducir el sistema de compresión en un sensor inteligente para evaluar su impacto.

5.1. Esquemático sensor

Se va a desarrollar un sensor inteligente compuesto por el acelerómetro BNO055 de Bosch y el microcontrolador Lopy4 en la Figura 36 se describe un diagrama de bloques del sensor inteligente.

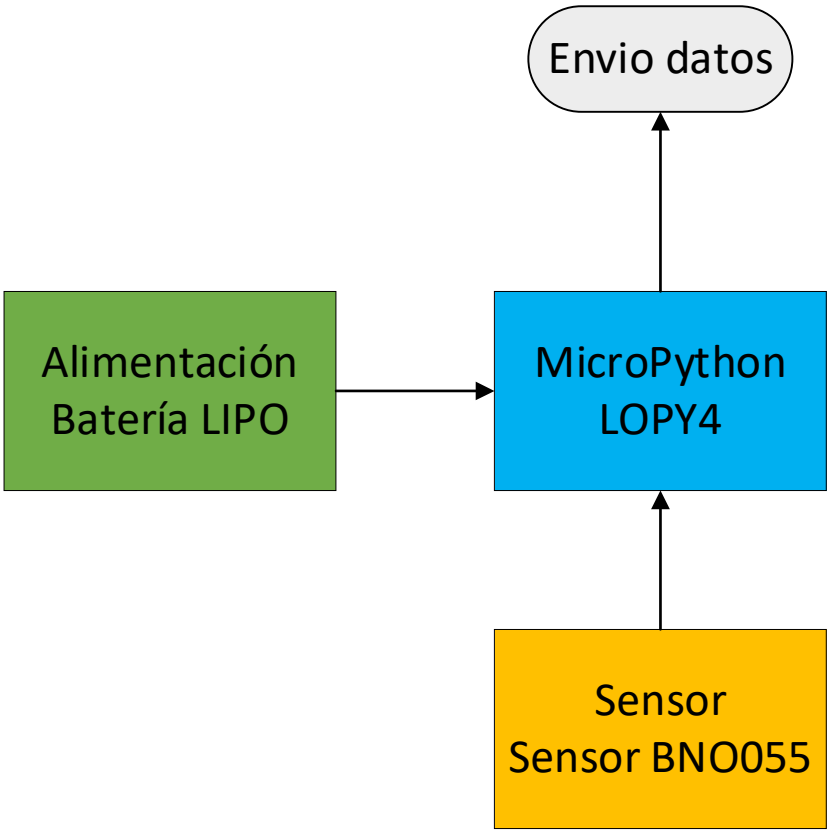


Figura 36.Diagrama bloques sensor

La lopy4 es un micro que cuenta con un procesador ESP32 capaz de correr micropython en su interior por lo que se va a adaptar el código desarrollado para poder correrlo dentro de este micro, cuenta con 4 MB de RAM y 8MB de ROM.
El micro acepta rangos de tensión de entre 3,5 y 5,5 V, que será alimentado por la batería LiPo de voltaje de celda 3,7 V, el sensor lo alimentaremos por la salida lógica de 3V3 de la lopy.

El micro contiene un módulo Wifi+BLE que nos permitirá realizar comunicación externa con el receptor para enviar los datos comprimidos.

5.2. Algoritmia

Se pretende implementar por un lado la parte de predicción del modelo compresor siguiendo un diagrama de flujo como el de la Figura 37, consiguiendo que el sensor genere las ventanas de datos para el sensor y pase estas por el codebook del modelo compresor, dando así la neurona ganadora.

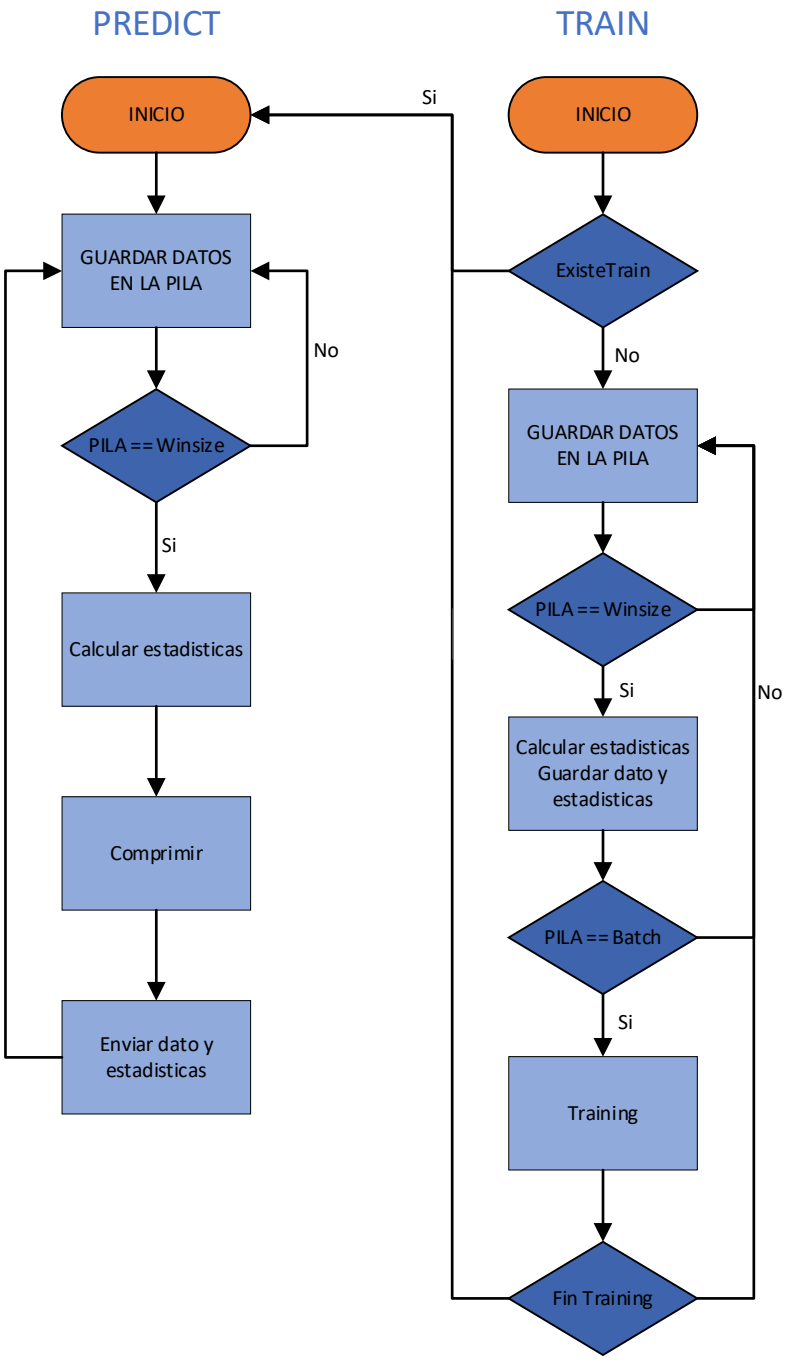


Figura 37.Diagrama Flujo predict y train micro

Con esto conseguiríamos enviar el dato comprimido en lugar del real aplicando la técnica de MSCL, sería necesario evaluar que configuración para los parámetros es la óptima, entrenando para distintas configuraciones como en el Capítulo 4 Análisis y resultados Modelo Vq.

En la Figura 37 se implementaría el modelo de entrenamiento para la red por medio de batches almacenados en la memoria del micro durante el tiempo de entrenamiento, estos datos vendrán condicionados por el tamaño del modelo VQ siendo necesaria una memoria superior al tamaño del codebook y de los propios datos de entrada.

Capítulo 6 Conclusiones

Por último, se hace un análisis del alcance del proyecto dividiendo en objetivos alcanzados y los pasos seguidos, así como las líneas futuras.

6.1. Objetivos alcanzados

En este proyecto se ha llevado a cabo un análisis de un sistema reductor de la dimensionalidad para la implementación en sensores inteligentes, se ha propuesto en paralelo un sistema de clasificación de estado para conseguir varios casos de uso, aunque visto el resultado del clasificador se descarta el método de entrenamiento de este como válido para este conjunto de datos.

Primero se ha llevado a cabo un estudio del estado del arte para comprobar que técnicas se usan habitualmente para sistemas de reducción de dimensionalidad, en este caso centrándose en los acelerómetros y la existencia de alguna base de datos que permitiera evaluar el sistema propuesto.

Como método alternativo a los modelos usuales de compresión se ha perseguido el desarrollo de un modelo de vector quantization adaptativo, definido en este caso para tres funciones objetivo.

Para llevar a cabo la evaluación de este método se ha desarrollado en lenguaje Python todo el modelo, depurando sus fallos hasta conseguir un funcionamiento correcto. Para la evaluación posterior del modelo se ha creado un código para realizar un entrenamiento automático que ejecute las variantes cargadas por un fichero json y vuelque los resultados.

En esta evaluación se han mezclado los parámetros de entrada de la red, con usuarios distintos y cogiendo varios sensores que, aunque del mismo fabricante y mismo usuario distaban de ser totalmente iguales, con el objetivo de ver el comportamiento del modelo a la generalización.

Se pretendió después caracterizar un clasificador de estado para codificar la tabla de neuronas en un número del 1 al 5, siendo este el total de actividades, de manera que para un caso de uso simplemente informativo consiguiéramos el menor tamaño dato de envío posible.

A través del Capítulo 4 Análisis y resultados apartado Conclusiones hemos podido corroborar el correcto funcionamiento del método para esta base de datos en concreto, obteniendo un resultado de compresión muy alto del orden de cientos de bits menos.

Este era el objetivo principal del proyecto para conseguir una implementación en los sistemas embebidos, para el caso de ejemplo se ha propuesto el desarrollo de este tipo de técnica en un Lopy4, que trabaja en microPython.

Implementado así el algoritmo compresor para tomar datos del sensor inteligente.

6.2. Líneas futuras

Realizar pruebas masivas para más bases de datos con más cantidad de muestras distintos perfiles para conseguir un modelo más amplio y reducir los casos no previstos por la red.

El desarrollo de estas técnicas en pequeños microcontroladores para ofrecer una vida útil prolongada frente al muestreo masivo al que ahora están sometidos.

Se podría seguir aumentando la reducción de dimensión pasando los parámetros media y desviación típica por otros modelos VQ para reducir estos tamaños.

Para un correcto comportamiento de clasificación se podría plantear otros modelos más destinados a estos menesteres como perceptrones multicapa o redes convolucionales.

Así como para brindar de mayor capacidad de reconstrucción a los datos entrenar un autoencoder que reconstruya los datos en función de los parámetros reducidos.

Referencias

References

[1] (2019). *Scikit-learn*. Available: https://scikit-learn.org/stable/index.html.

[2] (2019). *Pandas doc*. Available: https://pandas.pydata.org/.

[3] (2019). *Numpy doc*. Available: https://pandas.pydata.org/.

[4] F. Ganz *et al*, "A Practical Evaluation of Information Processing and Abstraction Techniques for the Internet of Things," *IEEE Internet of Things Journal,* vol. 2, *(4),* pp. 340-354, 2015.

[5] P. Prasertsung and T. Horanont, "A classification of accelerometer data to differentiate pedestrian state," *2016 International Computer Science and Engineering Conference (ICSEC),* pp. 1-5, 2016.

[6] M. H. M. Noor, Z. Salcic and K. I. Wang, "Adaptive sliding window segmentation for physical activity recognition using a single tri-axial accelerometer," *Pervasive and Mobile Computing,* vol. 38, pp. 41-59, 2017. . DOI: https://doi.org/10.1016/j.pmcj.2016.09.009.

[7] O. Banos *et al*, "Window size impact in human activity recognition," *Sensors (Basel, Switzerland),* vol. 14, *(4),* pp. 6474-6499, 04/09, 2014.

[8] Song-Mi Lee, Sang Min Yoon and Heeryon Cho, "Human activity recognition from accelerometer data using Convolutional Neural Network," *2017 IEEE International Conference on Big Data and Smart Computing (BigComp),* pp. 131-134, 2017.

[9] A. Jahanjoo, M. N. Tahan and M. J. Rashti, "Accurate fall detection using 3-axis accelerometer sensor and MLF algorithm," *2017 3rd International Conference on Pattern Recognition and Image Analysis (IPRIA),* pp. 90-95, 2017.

[10] N. C. Krishnan and S. Panchanathan, "Analysis of low resolution accelerometer data for continuous human activity recognition," *2008 IEEE International Conference on Acoustics, Speech and Signal Processing,* pp. 3337-3340, 2008.

[11] G. Marqués and K. Basterretxea, "Efficient Algorithms for Accelerometer-Based Wearable Hand Gesture Recognition Systems," *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing,* pp. 132-139, 2015.

[12] A. S. A. Sukor, A. Zakaria and N. A. Rahim, "Activity recognition using accelerometer sensor and machine learning classifiers," *2018 IEEE 14th International Colloquium on Signal Processing & its Applications (CSPA),* pp. 233-238, 2018.

[13] D. L. Padmaja and B. Vishnuvardhan, "Comparative study of feature subset selection methods for dimensionality reduction on scientific data," in *2016 IEEE 6th International Conference on Advanced Computing (IACC),* 2016, pp. 31-34.

[14] Z. Zhou, J. Mo and Y. Shi, "Data imputation and dimensionality reduction using deep learning in industrial data," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC),* 2017, pp. 2329-2333.

[15] P. Sancheti, R. Shedge and N. Pulgam, "Word-IPCA: An Improvement in Dimension Reduction Techniques," *2018 International Conference on Control, Power, Communication and Computing Technologies (ICCPCCT),* pp. 575-578, 2018.

[16] S. Choudhury *et al*, "Vector quantization and multi class support vector machines based fingerprint classification," *2016 International Conference on Inventive Computation Technologies (ICICT),* vol. 2, pp. 1-4, 2016.

[17] A. Banerjee and J. Ghosh, "Frequency-sensitive competitive learning for scalable balanced clustering on high-dimensional hyperspheres," *IEEE Transactions on Neural Networks,* vol. 15, *(3),* pp. 702-719, 2004.

[18] E. Pelayo, D. Buldain and C. Orrite, "Magnitude Sensitive Competitive Learning," *Neurocomputing,* vol. 112, pp. 4-18, 2013. . DOI: https://doi.org/10.1016/j.neucom.2012.11.039.

[19] (2019). *{UCI} Machine Learning Repository*. Available: http://archive.ics.uci.edu/ml. DOI: University of California, Irvine, School of Information and Computer Sciences.

[20] P. Casale, O. Pujol and P. Radeva, "Personalization and user verification in wearable systems using biometric walking patterns," *Personal and Ubiquitous Computing - PUC,* vol. 16, pp. 1-18, 06, 2012.

[21] K. Altun and B. Barshan, "Human activity recognition using inertial/magnetic sensor units," in *Hbu,* 2010, .

[22] A. Stisen *et al*, "Smart devices are different: Assessing and MitigatingMobile sensing heterogeneities for activity recognition," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems,* Seoul, South Korea, 2015, pp. 127-140.

[23] O. Banos *et al*, "mDurance: A Novel Mobile Health System to Support Trunk Endurance Assessment," *Sensors (Basel, Switzerland),* vol. 15, *(6),* pp. 13159-13183, 06/05, 2015.

[24] O. Banos *et al*, "Design, implementation and validation of a novel open framework for agile development of mobile health applications," *Biomedical Engineering Online,* vol. 14, *(S2:S6),* pp. 1-20, 2015.

[25] G. Sántha and G. Hermann, "Accelerometer based activity monitoring system for behavioural analysis of free-roaming animals," *2013 IEEE 11th International Symposium on Intelligent Systems and Informatics (SISY),* pp. 199-203, 2013.

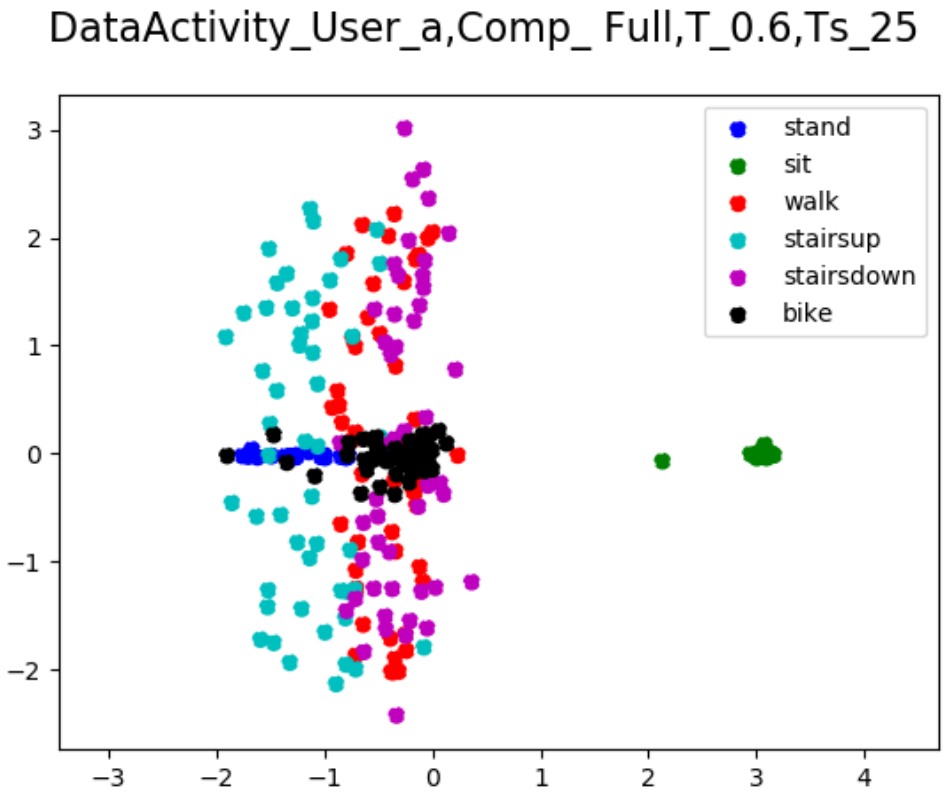
[26] C. Ladha *et al*, "Dog's life: Wearable activity recognition for dogs," in *UbiComp,* 2013, .

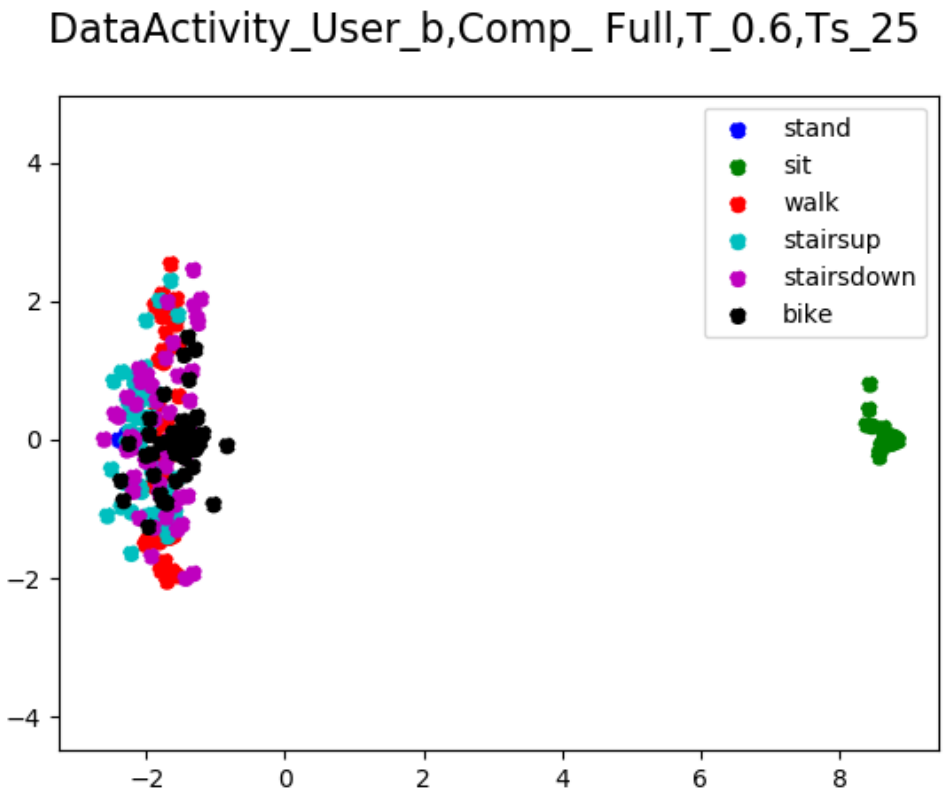
[27] L. Nóbrega *et al*, "Animal monitoring based on IoT technologies," *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany),* pp. 1-5, 2018.

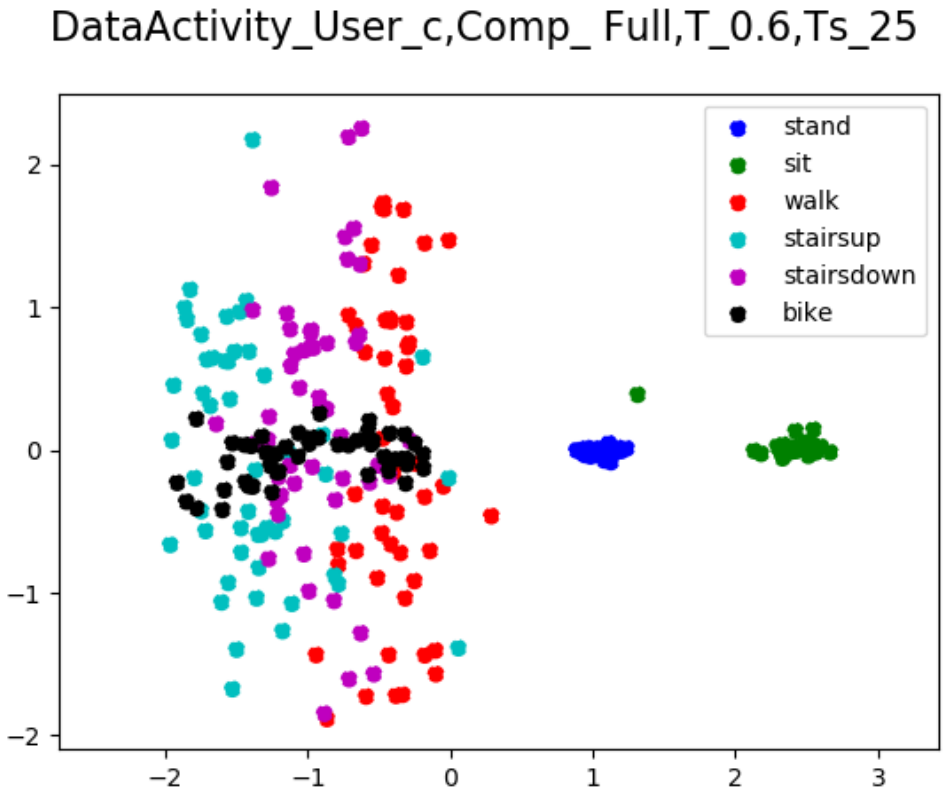
[28] A. Grami, "Chapter 5 - Analog-to-Digital Conversion," pp. 217-264, 2016. . DOI: https://doi.org/10.1016/B978-0-12-407682-2.00005-3.

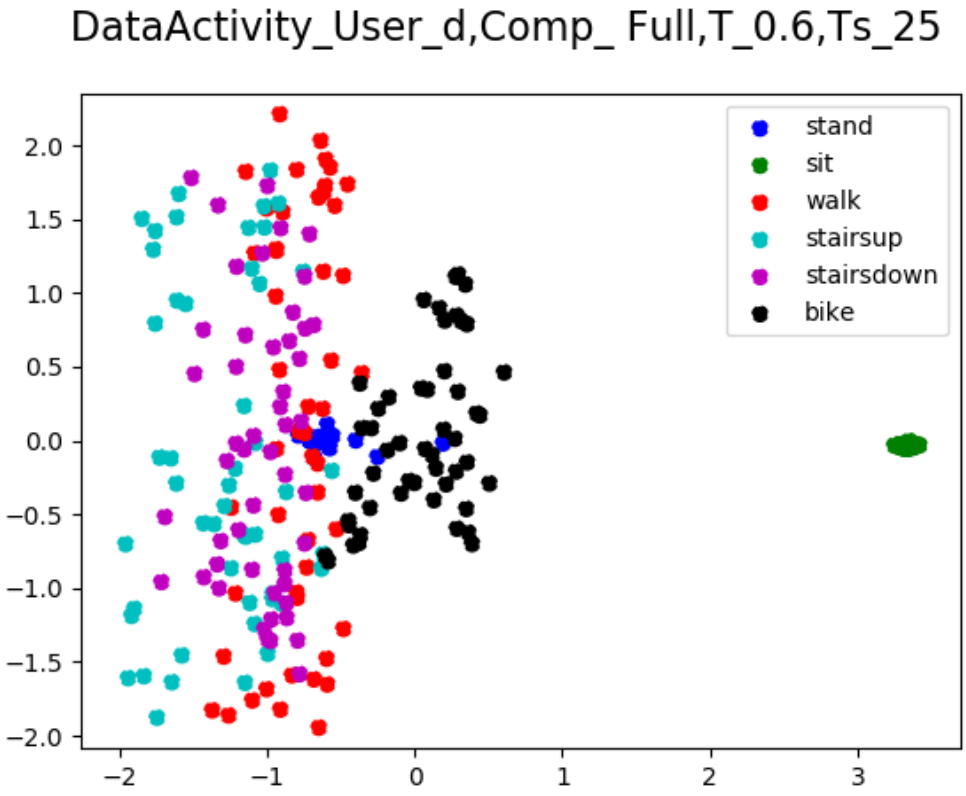
Anexos

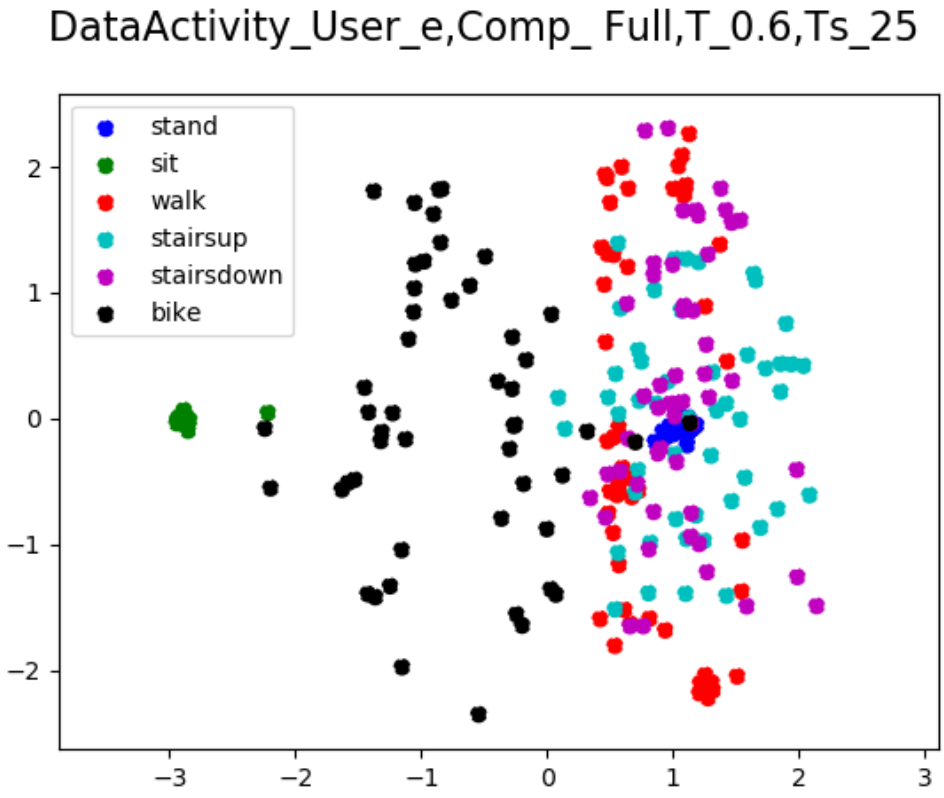
1. Análisis base de datos

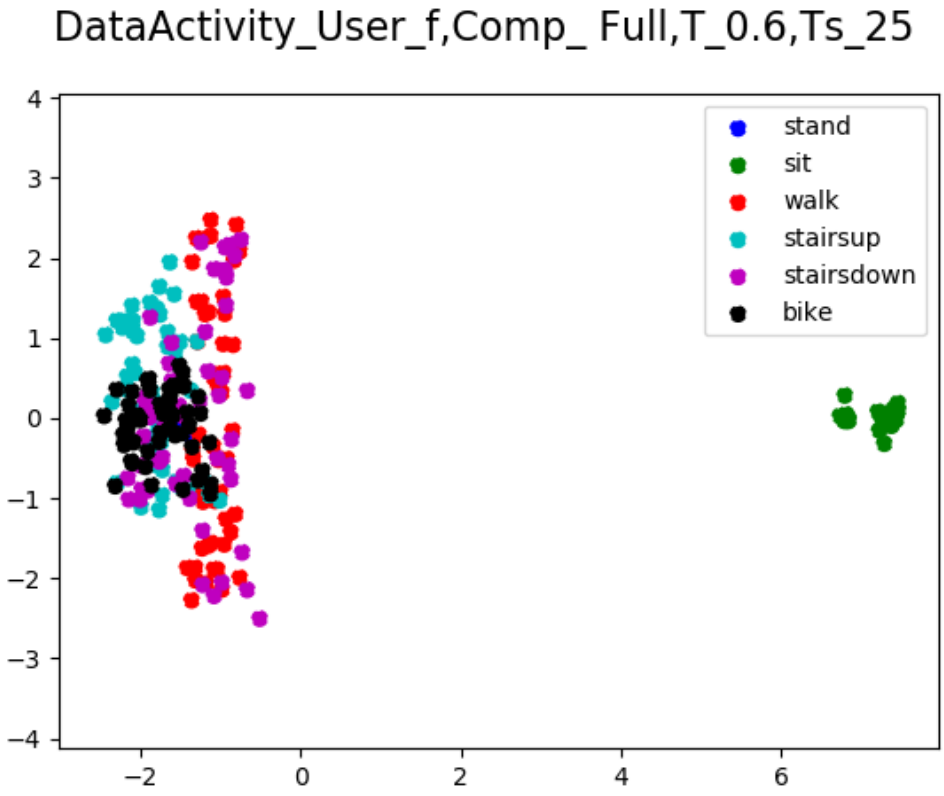


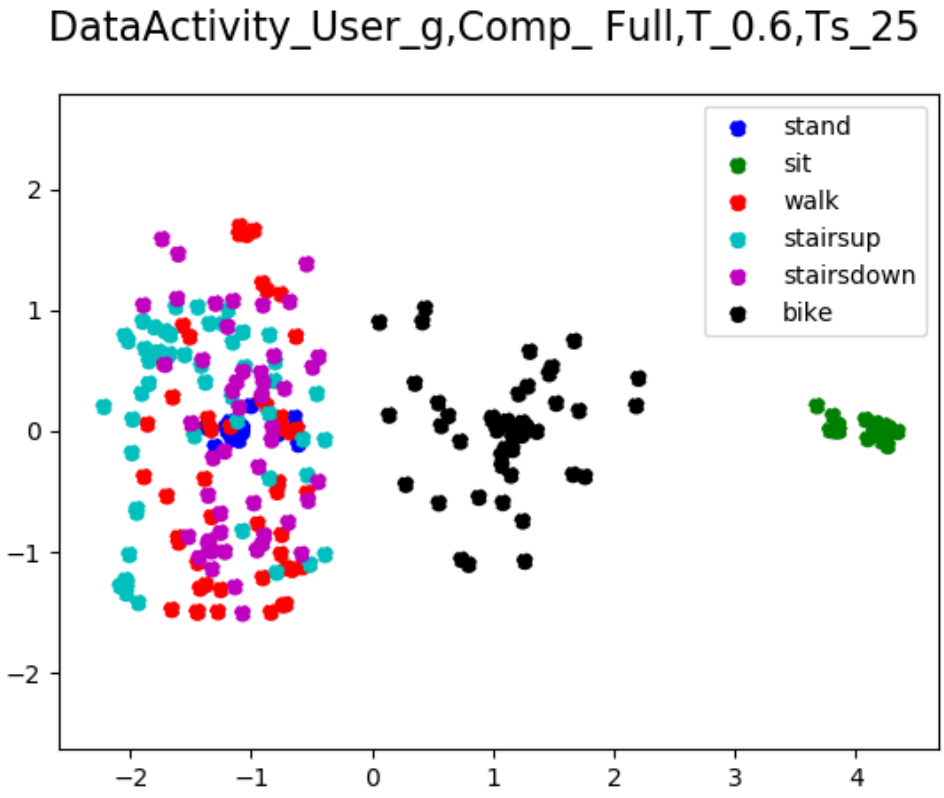


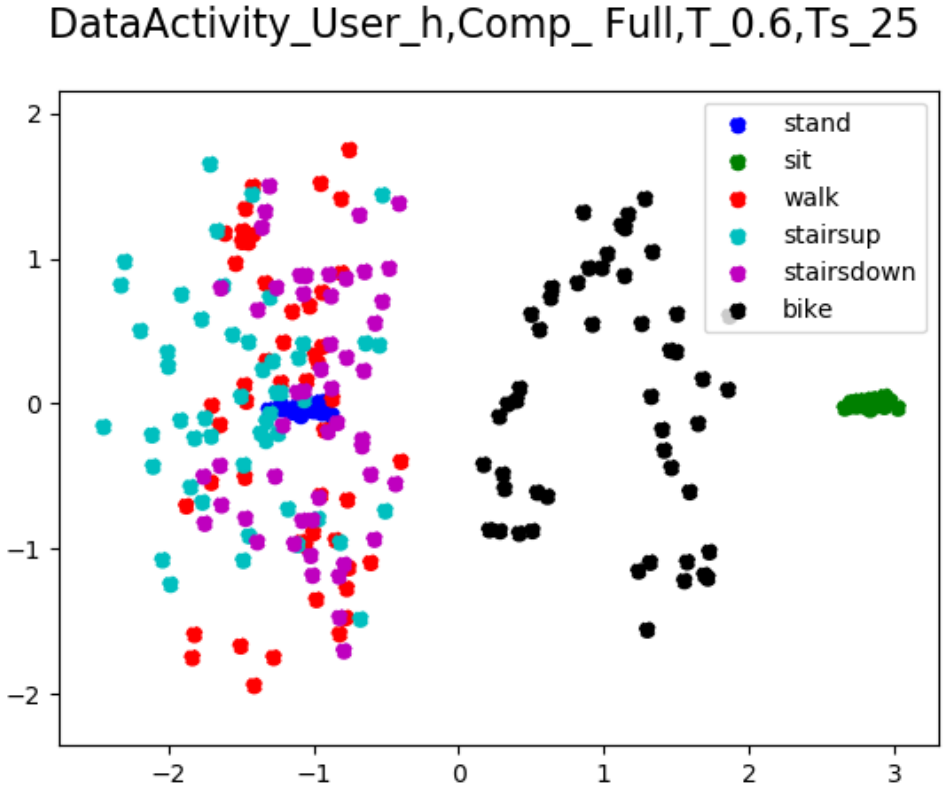

DataActivity_User_c,Comp_ Full,T_0.6,Ts_25
stand
sit
walk
stairsup
stairsdown
bike
2
1
0
−1
−2
−2
−1
0
1
2
3

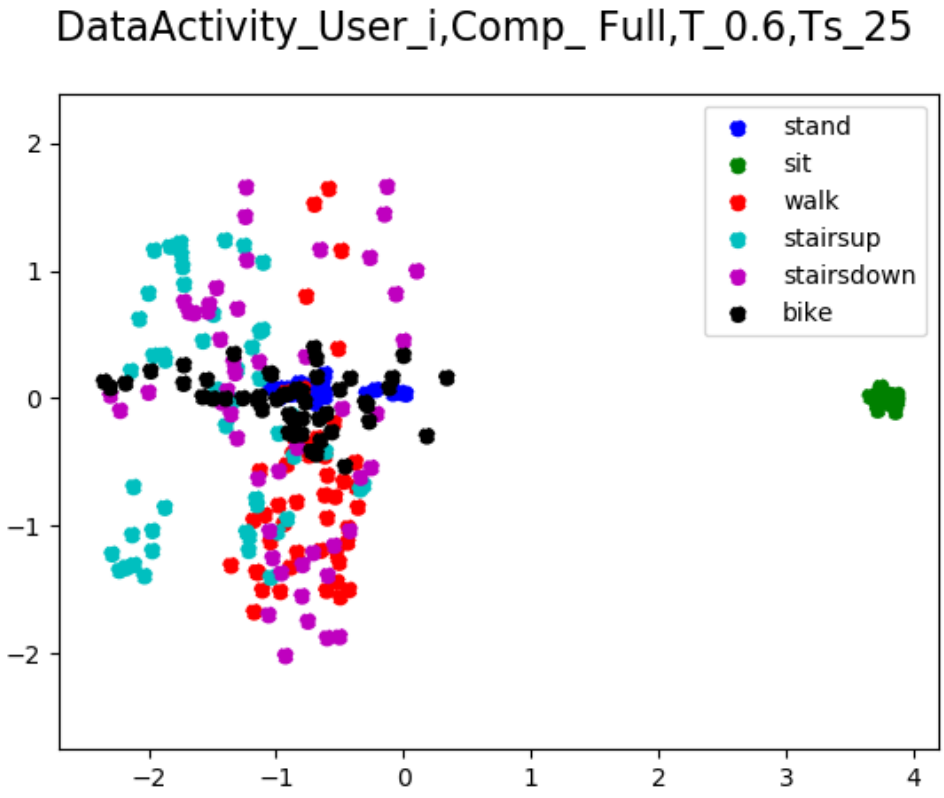

DataActivity_User_d,Comp_ Full,T_0.6,Ts_25
stand
sit
walk
stairsup
stairsdown
bike
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2
3


DataActivity_User_e,Comp_ Full,T_0.6,Ts_25
stand
sit
walk
stairsup
stairsdown
bike
2
1
0
−1
−2
−3
−2
−1
0
1
2
3


DataActivity_User_f,Comp_ Full,T_0.6,Ts_25
stand
sit
walk
stairsup
stairsdown
bike
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
6


DataActivity_User_g,Comp_ Full,T_0.6,Ts_25
stand
sit
walk
stairsup
stairsdown
bike
2
1
0
−1
−2
−2
−1
0
1
2
3
4


DataActivity_User_h,Comp_ Full,T_0.6,Ts_25
stand
sit
walk
stairsup
stairsdown
bike
2
1
0
−1
−2
−2
−1
0
1
2
3


DataActivity_User_i,Comp_ Full,T_0.6,Ts_25
stand
sit
walk
stairsup
stairsdown
bike
2
1
0
−1
−2
−2
−1
0
1
2
3
4

2. Modelo Vq

2.1. Fichero configuración

```
  "DataBase":{
   "Path":    "D:/TFM-ACTIVITY RECONITION/Activity recognition
exp/Phones_accelerometer.csv",
    "Users": ["a","b","c","d","e","f","g","h","i"],
    "UserTrain": "e",
    "UserTrain2": "f",
    "UserTrain3": "c",
    "SensorTrain": "s3",
    "SensorTest":  "s3",
    "AxisSensor":  ["x","y","z"],
    "LabelCols": ["gt"],
    "Activities": ["stand","sit","walk","stairsup","stairsdown","bike"],
    "Ts_sensor": 100,
    "SensorRange": 39.24,
    "Units": "ms2"
  },
  "PreProcess":{
    "Composition": ["Full","Single"],
    "Stadistics": [["Mean","Std"]],
    "Fs": [25,50,100],
    "T": [0.6,1,1.6]
  },
  "Compress_model": {
    "MSCL": {
      "Train%data": 0.65,
      "Validation%data": 0.20,
      "Test%data": 0.15,
      "Neurons":[1024,2048,4096,8192],
      "TargetFuntion": ["unity","FSCL","ESCL"],
      "Tepoch": 50,
      "Batch": 100,
      "Alphaini": 0.01,
      "Alphaend": 0.00001
      },
    "PCA": {
      "N_comp_plot": 2,
      "N_comp_evaluate": 10
    }
  },
  "Predict": {
    "path" :"C:/Users/darkj/Desktop/git/TFM_JORGE/Trains/Predict/"

  }
}
```

2.2. Algoritmo MSCL

```
"""
-------------------------------------------------------

=======================================================
MSCL algorithm
=======================================================

Autor: Jorge Vizarraga Turmo
```

```
    Version:5.1

    Function Reference
    --------------------------------------------------

    _init()

    random_weights(self, data_input)

    random_weights_1(self, data_input)

    initialize_train( self, data_length, inputs_number )

    get_bmu_nbmu(self, row_data)

    get_bmu(self, row_data)

    get_lbmu(self, bmu, nmu)

    refresh_Dw(self, data_input, datasample, lbmu, qerror_win, index_win,
    MatDW)

    calculate_alpha(self,alpha,alphaini,alphafin,epoch)

    adjust_weights(self, MatDW, alpha, index_win)

    initialize_magnitudetemp()

    update_magnitudetemp(self, mt, ma, bmu)

    update_magnitude(self, mt, ma,target_function)

    calculateShanon(self, neurons, totaldata, dataactivate)

    fit_model(self, X_train, X_validation, alphaini,
    alphaend,target_function="unity")

    predict(self, X_test)

    """

    import numpy as np
    import random
    import math
    import logging
    import time
    import datetime
    import matplotlib.pyplot as plt


    LIMIT_INITIAL_CENTROID = 1000
    ALPHA_POWER = 0.5
    Epsilon = 0.00001
    logging.basicConfig(format='%(asctime)s - %(message)s',level=logging.INFO)


    class MSCL:
```

```
    """
    Parameters:
        Neurons:Total of vecocts to compone the codebook
        Bacth_size: number of data input to divide the dataset
        Number_batch: number of batch per epoch
ceil(TotalInputdata/Batch_size)
        Number of epoch: times to repeat the process
        Magnitude vector: Magnitudes to evaluate the second iteration
between BMU and NMU


    """

    def __init__(self, neurons=0, batch_size=0, tcycles=0,Composition =
"Full"):

        self.weight = list()
        self.neurons = neurons
        self.magnitude_vector = list()
        self.batch_size = batch_size
        self.tcycles = tcycles
        self.Composition = Composition

    def random_weights(self, data_input):
        """
        Create Random weights for Neurons,weight are selected from
data_input

        ====================================
        Dimension = (Neurons*data_input_size)
        ====================================
        :param data_input: training data for take cols as weights (M)

        """
        self.weight = np.zeros( (self.neurons, data_input.shape[1])
).astype(np.float16)
        for i in range( len(self.weight) ):
            self.weight[i] = random.choice( data_input )

        self.magnitude_vector=np.ones( (self.neurons,1)
).astype(np.float16)

    def random_weights_1(self, data_input):
        """
        Create Random weights for Neurons,weight are selected from
data_input

        ====================================
        Dimension = (Neurons*data_input_size)
        ====================================
        :param data_input: training data for take cols as weights (M)

        """
        self.weight = np.zeros( (self.neurons, data_input.shape[1])
).astype(np.float16)

        if (data_input.shape[0] < LIMIT_INITIAL_CENTROID):
            mean_distribution = np.mean(data_input,
axis=0).astype(np.float16)
```

```
        else:
            data_input = np.random.permutation(data_input)
            mean_distribution =
np.mean(data_input[0:LIMIT_INITIAL_CENTROID - 1]).astype(np.float16)

        self.weight = mean_distribution + (5 *
np.random.random_sample((self.neurons, data_input.shape[1])))
        self.magnitude_vector=np.ones( (self.neurons,1)
).astype(np.float16)

    def initialize_train( self, data_length, inputs_number ):
        """

        :param data_length: Number of data_samples use to train
        :param inputs_number: Number of data_inputs (cols) have the train
sample
        :return: vector qerrorwin ,indexwin and MatDW initialize to zeros
        """
        qerror_win = np.zeros((data_length, 1)).astype(np.float16)
        index_win = np.zeros( (self.neurons, 1) ).astype(np.float16)
        MatDW = np.zeros( (self.neurons, inputs_number)
).astype(np.float16)

        return qerror_win, index_win, MatDW

    def get_bmu_nbmu(self, row_data):
        """

        :param row_data: Array 1D from DataInput
        :return: first [index,distance] and second BMU [index,distance]
        """

        qerrorlist = list()
        try:
            #Calculate Qerror
            data_input =
np.ones((self.neurons,row_data.shape[0])).astype(np.float16) *
np.reshape(row_data,(1,row_data.shape[0]))
            qerrorlist = (np.mean(( data_input - self.weight) ** 2
,axis=1)*self.weight.shape[1]).tolist()

            #Search BMU and NMU
            indexbmu = qerrorlist.index(min(qerrorlist))
            bmu = [(indexbmu,qerrorlist[indexbmu])]
            qerrorlist.pop(indexbmu)
            indexnmu = qerrorlist.index(min(qerrorlist))

            #Update indexBMU if is after bmu index increase 1 (pop delete
indexbmu)
            if ( ( indexnmu >= indexbmu ) and ( indexnmu <= (len(
qerrorlist ) - 1 ) ) ):
                nmu = [( indexnmu + 1, qerrorlist [indexnmu]) ]
            else:
                nmu = [( indexnmu , qerrorlist [indexnmu] )]

        except Exception as e:
            logging.fatal(e, exc_info=True)
```

```
            return bmu,nmu

        def get_bmu(self, row_data):
            """

            :param row_data: Array 1D from DataInput
            :return: first [index,distance]
            """

            qerrorlist = list()

            try:
                # Calculate Qerror
                data_input = np.ones((self.neurons,
row_data.shape[0])).astype(np.float16) * np.reshape(row_data, (1,
row_data.shape[0]))
                qerrorlist = (np.mean((data_input - self.weight) ** 2,
axis=1)*self.weight.shape[1]).tolist()
                # BMU index
                bmu = qerrorlist.index( min(qerrorlist))

            except Exception as e:
                logging.fatal(e, exc_info=True)

            return [(bmu, qerrorlist[bmu])]

        def get_lbmu(self, bmu, nmu):
            """

            :param bmu: Best neuron for the datainput sample
            :param nmu: Second neuron for the datainput sample
            :return: Lbmu the winner of qerror * magnitude_vector
[(index,querror,querror*M)]
            """
            qerrorM = list()
            indexlbmu = [bmu[0][0], nmu[0][0]]
            qerrorM.append(  (bmu[0][1] * self.magnitude_vector[bmu[0][0]] ) )
            qerrorM.append( (nmu[0][1] * self.magnitude_vector[nmu[0][0]] ))
            lbmu = qerrorM.index( min( qerrorM ) )

            return [( indexlbmu[lbmu], qerrorM[lbmu] )]

        def refresh_Dw(self, data_input, datasample, lbmu, qerror_win,
index_win, MatDW):
            """

            :param lbmu: Winner neuron
            :param index_win: vector for winning index for data_inputs
            :param qerror_win: vector for qerror winners for data_inputs
            :param MatDW: Mat to acumulate qerror and weights
            :return: Updated index_win, qerror_win, MatDW
            """
            qerror_win[datasample] = lbmu[0][1]
            index_win[lbmu[0][0]] += 1

            MatDW[lbmu[0][0]] += ( data_input - self.weight[lbmu[0][0]] )

            return qerror_win, index_win, MatDW
```

```
    def calculate_alpha(self,alpha,alphaini,alphafin,epoch):
        """
        Calculate the learning factor forced to decay with iteration time
        :param alpha: Actual value for alpha (start in alphaini)
        :param alphaini: constant value for alpha in t=0
        :param alphafin: constant valua for alpha expected in t=T (number
of epoch)
        :param epoch: Number of epoch t
        :return: alpha (t)
        """
        try:
            #alpha highstart
            #alpha = math.pow( ( alphafin * ( alphaini / alpha) ), (epoch
/ self.tcycles) )

            #alpha softstart
            alpha = (alphaini + (alphafin - alphaini) * math.pow((epoch /
self.tcycles), ALPHA_POWER))

        except Exception as e:
            logging.fatal( e, exc_info=True)
        return alpha

    def adjust_weights(self, MatDW, alpha, index_win):
        """
        Adjust weights for neurons to next cycle follow this ecuation:

        ========================================================
        Weights(t+1)=weights(t)+alpha*(data_iputs - weights_Lbmu)
        ========================================================

        :param MatDW: Codebook with acumulation (data_input-weightLBMU)
        :param alpha: learning factor calculate for t epoch
        :return:
        """
        try:
            index_nondivisible = index_win + Epsilon # +Epsilon ZERO
division
            AcumulationSum = ( alpha * ( MatDW / (index_nondivisible)
)).astype(np.float16)
            self.weight = self.weight + AcumulationSum


        except Exception as e:
            logging.fatal( e, exc_info=True)


    def initialize_magnitudetemp(self):
        """
        Inicializar magnitude temp vector to evalue next magnitude
        :return: Mt , Ma vector zeros magnitude_vector size
        """
        M_size = len( self.magnitude_vector )
        Mt = np.zeros( (M_size,1) ).astype(np.float16)
        Ma = np.zeros( (M_size,1) ).astype(np.float16)

        return Mt, Ma

    def update_magnitudetemp(self, mt, ma, bmu):
```

```
        """
        We accumulate in the Mt vectors the value of the BMU Querror and
Ma an activation for the BMU index

        :param mt: Accumulate Querror of BMU'S
        :param ma: Number of winning for this neurons
        :param bmu: Winner neuron
        :return:
        """
        mt[bmu[0][0]] += bmu[0][1]
        ma[bmu[0][0]] += 1

        return mt,ma


    def update_magnitude(self, mt, ma,target_function):
        """
        Calculate new magnitude vector depend of target_function
        :param mt: BMU Magnitude temporal
        :param ma: BMU Activation temporal
        :param target_function:
        :return: Update vector M value for next epoch
        """
        M_size = len(self.magnitude_vector)
        magnitude = np.zeros((M_size, 1)).astype(np.float16)

        if target_function == "unity" :
            magnitude = self.magnitude_vector

        if target_function == "FSCL" :
            magnitude = ma

        if target_function == "ESCL":

           # for i in range(M_size):

            #    if ma[i] != 0 :
            magnitude = mt / (ma +Epsilon) # +Epsilon ZERO division

        return magnitude.astype(np.float16)

    def calculateShanon(self, neurons, totaldata, dataactivate):
        """
        Calculate Shannon entropy
        :param neurons: number of neruons
        :param totaldata: total data samples
        :param dataactivate: index wining bmu

        """
        SumatorioPH = 0
        #Probability winindex
        Pi = dataactivate / totaldata
        #Max Shanon
        MaxH = (1 / neurons) * (math.log(neurons))
        #Sum_probability
        for i in Pi:
            if (i == 0):
                pass
            else:
```

```
                SumatorioPH += i * (math.log(i))
            #Search null neurons
            Nullneurons = [np.where(Pi == 0.0)]
            Contnullneurons = len(Nullneurons) - 1
            H = (-1 / neurons * SumatorioPH) / MaxH

            return H, Nullneurons, Contnullneurons

        def fit_model(self, X_train, X_validation, alphaini,
    alphaend,tarjet_function="unity"):
            global weights
            """
             Fit model with the paramaters

            :param X_train: Train data Array 2d
            :param X_validation: Validation data Array 2d
            :param alphaini: Learning factor ini
            :param alphaend: Learning factor end
            :return: model fit
            """
            cont_ciclos = 0
            cont_batches = 0
            cont_datos = 0
            alpha = alphaini
            weights=np.zeros((self.neurons,1)).astype(np.float16)

            self.random_weights(X_train)

            vecalpha = np.array(([alpha]))
            vecmagmedio = np.array(([np.mean(self.magnitude_vector)]))
            vecshanon = np.zeros((1))
            vecnullneurons = np.zeros((1))
            vecmt = np.array(([np.mean(self.magnitude_vector)]))
            vecmtv = np.array(([np.mean(self.magnitude_vector)]))
            Maacum = np.zeros( (len( self.magnitude_vector ),1) )
            Mavacum = np.zeros( (len( self.magnitude_vector ),1) )
            while cont_ciclos != self.tcycles:
                #Init conts
                cont_batches = 0
                cont_datos = 0
                #Calculate weights
                qerror_win, index_win, MatDW = self.initialize_train(
    X_train.shape[0], X_train.shape[1] )
                Tb = ( math.ceil( X_train.shape[0] / self.batch_size ) )

                while cont_batches != Tb :

                    while (cont_datos != self.batch_size) and (cont_datos + (
    cont_batches * self.batch_size ) <= (X_train.shape[0])-1) :
                        data_sample = cont_datos + ( cont_batches *
    self.batch_size )
                        bmu,nmu = self.get_bmu_nbmu( X_train[data_sample] )
                        lbmu = self.get_lbmu( bmu, nmu )
                        qerror_win, index_win, MatDW = self.refresh_Dw(
    X_train[data_sample], data_sample, lbmu, qerror_win, index_win, MatDW)
                        cont_datos += 1

                    #print("Batch:"+ str(cont_batches))
                    cont_batches+=1
```

```
                cont_datos=0
            cont_batches = 0
            cont_datos = 0
            cont_ciclos += 1


            #Update weights
            alpha = self.calculate_alpha( alpha, alphaini, alphaend,
cont_ciclos )

            vecalpha = np.concatenate((vecalpha, np.array(([alpha]))),
axis=0)
            self.adjust_weights( MatDW, alpha, index_win )

            #Representacion Adjust weights
            weights = np.concatenate((weights,self.weight),axis=1)

            #Start Magnitude update
            Mt, Ma = self.initialize_magnitudetemp()
            while cont_batches != Tb:

                while (cont_datos != self.batch_size) and (cont_datos + (
cont_batches * self.batch_size ) <= (X_train.shape[0])-1)  :
                    data_sample = cont_datos + ( cont_batches *
self.batch_size)
                    bmu= self.get_bmu(X_train[data_sample])
                    Mt, Ma = self.update_magnitudetemp( Mt, Ma, bmu)
                    cont_datos += 1
                cont_batches += 1
                cont_datos = 0


            #update Magnitude per epoch
            self.magnitude_vector =
self.update_magnitude(Mt,Ma,tarjet_function)

            #Validation Evaluation
            cont_batches = 0
            cont_datos = 0
            Mtv, Mav = self.initialize_magnitudetemp()
            Tb=( math.ceil( X_validation.shape[0] / self.batch_size ) ) -1
            while cont_batches != Tb:

                while (cont_datos != self.batch_size) and (cont_datos + (
cont_batches * self.batch_size ) <= (X_validation.shape[0])-1)  :
                    data_sample = cont_datos + ( cont_batches *
self.batch_size )
                    bmu = self.get_bmu( X_validation[data_sample])
                    Mtv, Mav = self.update_magnitudetemp(Mtv, Mav, bmu)
                    cont_datos += 1

                cont_batches += 1
                cont_datos = 0

            if (cont_ciclos % np.floor(self.tcycles / 10) == 0):
                print("Epoch:" + str(cont_ciclos))
                # print("Train_Magnitude_mean:" + str(np.mean(self.Mt)))
                print("Tra_Querr_mean:" + str(np.mean(Mt)))
                print("Tral_Activ_mean:" + str(np.mean(Ma)))
```

```
                print("Eval_Querr_mean:" + str(np.mean(Mtv)))
                print("Eval_Activ_mean:" + str(np.mean(Mav)))

            vecmagmedio = np.concatenate((vecmagmedio,
np.array(([np.mean(self.magnitude_vector)]))), axis=0)
            # Shannon_Entropy
            ShanonEntropy, nullneurons, Contnullneurons =
self.calculateShanon(self.neurons, X_train.shape[0], Ma)
            vecshanon = np.concatenate((vecshanon, ShanonEntropy), axis=0)
            vecnullneurons = np.concatenate((vecnullneurons,
np.array(([Contnullneurons]))), axis=0)

            vecmt = np.concatenate((vecmt, np.array(([np.mean(Mt)]))),
axis=0)
            vecmtv = np.concatenate((vecmtv, np.array(([np.mean(Mtv)]))),
axis=0)
            Maacum += Ma
            Mavacum += Mav

        date =
datetime.datetime.fromtimestamp(time.time()).strftime('%Y%m%d_%H%M%S')
        fig, axs = plt.subplots(4, 1)
        fig.tight_layout()
        axs[0].plot(np.array(range(self.tcycles + 1)), vecalpha)
        axs[0].set_title('Alpha vs Tcycles')
        axs[1].plot(np.array(range(self.tcycles + 1 )), vecmagmedio)
        axs[1].set_title('MagnitudeMean vs Tcycles')
        axs[2].plot(np.array(range(self.tcycles + 1)), vecshanon)
        axs[2].set_title('ShanonEntropy vs Tcycles')
        axs[3].plot(np.array(range(self.tcycles + 1)), vecnullneurons)
        axs[3].set_title('Nullneurons vs Tcycles')

plt.savefig("Plots/Trainings/MSCL/ParamsCompare_Neurons_"+str(self.neurons
)+",Time"+str(date)+",TF_"+str(tarjet_function)
+",Composition_"+str(self.Composition)+  '.png')


        fig2, axs2 = plt.subplots(4, 1)
        fig2.tight_layout()
        axs2[0].plot(np.array(range(self.tcycles + 1 )), vecmt)
        axs2[0].set_title('Tra_Querr_mean_Accumulated')
        axs2[1].plot(np.array(range(self.tcycles + 1 )), vecmtv)
        axs2[1].set_title('Eval_Querr_Accumulated')
        axs2[2].plot(np.array(range(self.neurons)),Maacum /
self.tcycles,c=np.random.rand(3,))
        axs2[2].set_title('Tra_Activ_Accumulated')
        axs2[3].plot(np.array(range(self.neurons)),Mavacum /
self.tcycles,c=np.random.rand(3,))
        axs2[3].set_title('Eval_Activ_mean_Accumulated')

plt.savefig("Plots/Trainings/MSCL/ParamsAccumulated_Neurons_"+str(self.neu
rons)+",Time"+str(date)+",TF_"+str(tarjet_function)
+",Composition_"+str(self.Composition)+ '.png')

        plt.close('all')

    def predict(self, X_test):
        """
        Return predicted values from input data 2d
```

```
            :param X_test: Test data array 2d
            :return:
            """

            cont_datos = 0
            qerror_winner = np.zeros( ( X_test.shape[0], 1)
    ).astype(np.float16)
            index_winner = np.zeros( (X_test.shape[0], 1) ).astype(np.float16)
            Matpredict = np.zeros( (X_test.shape[0], self.weight.shape[1])
    ).astype(np.float16)
            while cont_datos != X_test.shape[0]:
                data_sample = cont_datos
                bmu = self.get_bmu( X_test[data_sample] )
                index_winner[data_sample] = bmu[0][0]
                qerror_winner[data_sample] = bmu[0][1]
                Matpredict[data_sample] =  self.weight[bmu[0][0]]
                cont_datos += 1



            return Matpredict, qerror_winner, index_winner.astype(int)
```

2.3. Script entrenamiento automático

```
"""
-------------------------------------------------------

=======================================================
Data_Base Train
=======================================================
Autor: Jorge Vizarraga Turmo

Date: 15/10/2019

Version:1.0

Function Reference
-------------------------------------------------------

"""
import numpy as np
import pickle
import datetime
import pandas as pd
import time
import json
import logging
import pymscl
from sklearn import decomposition
from itertools import cycle
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split


#Global parametres
configjson = "config" #Fichero Json con las opciones de pruebas
NMN=50 #Numero de muestras por clase para Plot
NBSTD = 32 #Numero de bits para la estaditica
EPSILON = 0.00001

def searchJson(file, search):
    """
    Function to search into JSON giving a file.json and the search sequence
    :param file: path with the name of the file
    :param search: search sequence in list [search1,search2,...,searchn]
    :return: search result
    """
    try:
        jsonread = json.loads(open(file + '.json').read())
    except Exception as e:
        logging.error("Exception occurred", exc_info=True)
        result = ""
    if (len(search) < 2):
        result = jsonread.get(search[0])
    else:
        result = jsonread.get(search[0])

    for i in range(len(search) - 1):
        result = result.get(search[i + 1])

    return result
```

```
def LoadMat(path, sensor, user, cols, labelcol):
    """
    Function to load the dataset into a numpy mat, giving pathfile, sensor,
user and
    index of sensor params you want into a list.
    :param path: path file csv
    :param sensor: device you want to use
    :param user: user you want to use
    :param cols: sensors cols into a list
    :return: mat numpys , mat labels
    """

    try:
        log = pd.read_csv(path, sep=",")
        device = log.where(log["Model"] == sensor)
        device = device.dropna()
        userdevice = device.where(device["User"] == user)
        userdevice = userdevice.dropna()

    except Exception as e:
        logging.error("Exception occurred", exc_info=True)

    try:
        Matdevice = userdevice[cols]
        MatLabels = userdevice[labelcol]
    except Exception as e:
        logging.error("Exception occurred", exc_info=True)
        Matdevice = np.zeros((1, 1))
        MatLabels = np.zeros((1, 1))

    return Matdevice.to_numpy().astype(np.float16), MatLabels.to_numpy()


def CalculateWindow(Mat, winsize, overlap_count=0):
    """
    Function to divide the dataset into windows with a winsize and if you
want to
    make overlap use overlap_count in funtion of data you want.
    :param Mat: data input
    :param winsize: number of data for window
    :param overlap_count: number of data for time scroll
    return: Window_mat
    """

    i = 0
    j = 0
    timeini = time.time()

    # NaN division Zero
    if (overlap_count == 0):
        filas = (int(Mat.shape[0] / winsize))
    else:
        filas = (round((Mat.shape[0] - (winsize - overlap_count)) /
overlap_count))

    Matwindow = np.zeros((filas, winsize), dtype=Mat.dtype)
    timeini2 = time.time()
```

```
    while (i <= (Mat.shape[0] - winsize)):
        try:
            aux = np.transpose(Mat[i:(i + winsize)])
        except Exception as e:
            logging.error("Exception occurred", exc_info=True)

        if (overlap_count == 0):
            i += winsize
        else:
            i += overlap_count

        try:
            Matwindow[j] = aux
            j += 1

        except Exception as e:
            logging.error("Exception occurred", exc_info=True)

    logging.info("Matriz Dimension:" + str(Matwindow.shape) +
                 " Timecompletewindow(s): " + str((time.time() - timeini)))

    return Matwindow


def CalculateStadistics(Mat, vectindicators):
    """
    Funtion to extract indicators : ["Mean", "Median", "Max", "Min",
"Standar_Desviation",
    "Variance","Derived","Correlation"]

    :param Mat: Mat numpy
    :param vectindicators:list of indicators select one or more of those on
the list
    :return:Mat with indicators calculated and Matindicators only
    """
    timeini = time.time()
    filas = Mat.shape[0]
    windowsize = Mat.shape[1]
    Matindicators = np.zeros((filas, 1))
    try:
        if ("Mean" in vectindicators):
            Matindicators = np.append(Matindicators, np.mean(Mat,
axis=1).reshape(filas, 1), axis=1)

        if ("Median" in vectindicators):
            Matindicators = np.append(Matindicators, np.median(Mat,
axis=1).reshape(filas, 1), axis=1)

        if ("Max" in vectindicators):
            Matindicators = np.append(Matindicators, np.amax(Mat,
axis=1).reshape(filas, 1), axis=1)

        if ("Min" in vectindicators):
            Matindicators = np.append(Matindicators, np.amin(Mat,
axis=1).reshape(filas, 1), axis=1)

        if ("Std" in vectindicators):
            Matindicators = np.append(Matindicators, np.std(Mat,
```

```
axis=1).reshape(filas, 1), axis=1)

        if ("Variance" in vectindicators):
            Matindicators = np.append(Matindicators, np.var(Mat,
axis=1).reshape(filas, 1), axis=1)

        if ("Derived" in vectindicators):
            Matindicators = np.append(Matindicators, np.diff(Mat,
n=windowsize - 1, axis=1).reshape(filas, 1), axis=1)
    except Exception as e:
        logging.error("Exception occurred", exc_info=True)

    if ("Correlation" in vectindicators):
        numcorr = 3
        corrxy = 0
        corrxz = 0
        corryz = 0
        Matcorr = np.zeros((filas, numcorr))
        for i in range(Mat.shape[0]):
            veccorr = list()
            if (i % numcorr == 0):
                try:
                    corrxy = np.correlate(Mat[i], Mat[i + 1])
                    veccorr.append(corrxy)
                except:
                    corrxy = 0
                    veccorr.append(corrxy)
                try:
                    corrxz = np.correlate(Mat[i], Mat[i + 2])
                    veccorr.append(corrxz)
                except:
                    corrxz = 0
                    veccorr.append(corrxz)
                try:
                    corryz = np.correlate(Mat[i + 1], Mat[i + 2])
                    veccorr.append(corryz)
                except:
                    corryz = 0
                    veccorr.append(corryz)
            else:
                veccorr.append(corrxy)
                veccorr.append(corrxz)
                veccorr.append(corryz)
            try:
                Matcorr[i, :] = np.array(veccorr).reshape(1, numcorr)
            except Exception as e:
                logging.error("Exception occurred", exc_info=True)

        try:
            Matindicators = np.append(Matindicators, Matcorr, axis=1)
            del (Matcorr)

        except Exception as e:
            logging.error("Exception occurred", exc_info=True)

    Matindicators = np.delete(Matindicators, 0, axis=1)

    logging.info("Matriz Dimension:" + str(Matindicators.shape) +
                 " TimeCalculateIndicators(s): " + str(time.time() -
```

```
timeini))

    return Matindicators.astype(np.float16)


def MatDatabase(configjson, user):
    """
    Function to build Numpy array 2d from file csv, build with config
database
    params and scaled
    :configjson: path of the config json file
    :return: sensor array scaled
    """
    # Database Config
    path = searchJson(configjson, ['DataBase', 'Path'])
    SensorTrain = searchJson(configjson, ['DataBase', 'SensorTrain'])
    AxisSensor = searchJson(configjson, ['DataBase', 'AxisSensor'])
    Labels = searchJson(configjson, ['DataBase', 'LabelCols'])

    # Process Extract Database
    MatTrain, MatLabels = LoadMat(path, SensorTrain, user, AxisSensor,
Labels)

    return MatTrain, MatLabels


def plotDatavsLabel2D(configjson,MatData,path,namesave,MatLabel):
    """

    :param configjson: Json config
    :param MatData: data array2d
    :param path: path to save fig
    :param namesave: String with the save information
    :param MatLabel: Label array2d
    :return:
    """
    Activities = searchJson(configjson, ['DataBase', 'Activities'])
    cycol = cycle('bgrcmk') #gamma de colores

    figsubplots = plt.figure()
    plt.axis('equal')
    for i in Activities:
        Labels = np.where(MatLabel == i)
        Activity = MatData[Labels[0]]
        plt.scatter(Activity[0:NMN,0],Activity[0:NMN,1],
color=next(cycol),linestyle='dashed', label=str(i))
    try:
        plt.legend()
    except:
        pass
    figsubplots.suptitle('DataActivity'+namesave, fontsize=16)
    plt.savefig("Plots/"+path+ namesave + '.png')


def plotDatavsLabel3D(configjson,MatData,path,namesave,MatLabel):
    """

    :param configjson: Json config
    :param MatData: data array2d
```

```
    :param path: path to save fig
    :param namesave: String with the save information
    :param MatLabel: Label array2d
    :return:
    """
    Activities = searchJson(configjson, ['DataBase', 'Activities'])
    cycol = cycle('bgrcmk')

    figsubplots = plt.figure()
    ax = figsubplots.add_subplot(111, projection='3d')

    for i in Activities:
        Labels = np.where(MatLabel == i)
        Activity = MatData[Labels[0]]
        ax.scatter(Activity[0:NMN,0],Activity[0:NMN,1],Activity[0:NMN,2],
color=next(cycol),linestyle='dashed',label=str(i))

    try:
        plt.legend()
    except:
        pass
    figsubplots.suptitle('DataActivity '+namesave, fontsize=16)
    plt.savefig("Plots/"+path+namesave+ '.png')


def trainDataVQ(Mat, MatLabel, user, configjson):
    """
    * Select composition for components.

    :param Mat: Mat  from Dataset
    :param configjson: config json
    :param MatLabel: Label from dataset
    :param user: User from dataset
    """
    timeini = time.time()
    Composition = searchJson(configjson, ['PreProcess', 'Composition'])

    if ("Full" in Composition):
        trainwindows(Mat, MatLabel, "Full", user, configjson)
    if ("Single" in Composition):
        for i in range(Mat.shape[1]):
            trainwindows(np.reshape(Mat[:, i],(Mat.shape[0],1)), MatLabel,
"Single" + str(i), user, configjson)

    logging.info("Time Total TrainingProcess: " + str(time.time() - timeini)
+ " (s)")


def Apply_Stadistics(Mat,Stadistics,savename):
    """
    Apply Stadisitics Mean and Std to matData
    :param Mat: Data array 2d
    :param Stadistics: Data array 2d
    :param savename: String savename information
    :return: Matstadistics applied ,MatStadistics
    """

    # CalculeEstadistics
    for i in Stadistics:
```

```
        MatStadistics = CalculateStadistics(Mat, i)
        # Apply Mean
        Mat = Mat - np.reshape(MatStadistics[:, 0], (MatStadistics.shape[0],
1))
        # Apply Std
        Mat = Mat / np.reshape(MatStadistics[:, 1], (MatStadistics.shape[0],
1))

        # Save Stadistics
        try:
            np.save('Trains/Stadistics/Stadistics_' +savename + ".npy",
                    MatStadistics)
        except:
            print("Error saving MatStadistics")

    return Mat,MatStadistics


def Divide_index_Mat(Mat):
    """
    Get indexpar mat and indeximpar mat
    :param Mat: data input 2d
    :return: MatPar,MatImpar
    """

    par_index = np.array(list(range(0, Mat.shape[0], 2)))
    MatPar = Mat[par_index,:]
    MatImpar = np.delete(Mat, par_index, axis=0)

    return MatPar,MatImpar


def Windows_Maker(Mat,Composition,WindowSize):
    """
    Create windows data from array 2d with size = windowsize  and
concatenation depend of the composition
    :param Mat: data input 2d
    :param Composition: number of axis from sensor
    :param WindowSize: size cols data input
    :return: Mat windows
    """

    if (Composition == "Full"):
        for i in range(Mat.shape[1]):
            MatWin = CalculateWindow(Mat[:, i], WindowSize)

            if i == 0:
                MatWindow = MatWin
            else:
                MatWindow = np.concatenate((MatWindow, MatWin), axis=1)

        del (MatWin)

    else:
        MatWindow = CalculateWindow(Mat, WindowSize)

    return MatWindow
```

```
def Submuestreo(Matriz,Composition,WindowSize,fs,fs_sensor):
    """

    :param Matriz: data 2d
    :param Composition: number of axis from sensor
    :param WindowSize: size data cols
    :param fs: new freq
    :param fs_sensor: freq sensor
    :return: new mat
    """
    MatsPar = list()
    MatsImpar = list()
    MatPar, MatImpar = Divide_index_Mat(Matriz)
    MatsPar.append(MatPar)
    MatsImpar.append(MatImpar)
    numevals = math.floor(fs_sensor / fs)# times fsensor division

    if(numevals/2 != 1): #subsampling up 50%

        for i in range(int(numevals/2) - 1):#up 50% and eliminate 1st
conversion
            MatPar,MatImpar = Divide_index_Mat(MatsPar[i])
            MatsPar.append(MatPar)
            MatsImpar.append(MatImpar)

            MatPar,MatImpar = Divide_index_Mat(MatsImpar[i])
            MatsPar.append(MatPar)
            MatsImpar.append(MatImpar)
        del(MatPar)
        del(MatImpar)

        MatParwin = Windows_Maker(MatsPar[len(MatsPar)- int(numevals/2) ],
Composition, WindowSize)
        MatImParwin =  Windows_Maker(MatsImpar[len(MatsImpar)-
int(numevals/2) ], Composition, WindowSize)

        for i in range(int(numevals/2) - 1  ):
            MatParwin_t = Windows_Maker(MatsPar[len(MatsPar)-
(int(numevals/2) -1 +i) ], Composition, WindowSize)
            MatParwin = np.concatenate((MatParwin,MatParwin_t),axis=0)
            MatImParwin_t = Windows_Maker(MatsImpar[len(MatsImpar)-
(int(numevals/2) -1 +i)], Composition, WindowSize)
            MatImParwin = np.concatenate((MatImParwin, MatImParwin_t),
axis=0)
        del(MatParwin_t)
        del(MatImParwin_t)
    else:
        MatParwin = Windows_Maker(MatsPar[len(MatsPar)-1], Composition,
WindowSize)
        MatImParwin = Windows_Maker(MatsImpar[len(MatsImpar)-1], Composition,
WindowSize)

    return np.concatenate((MatParwin,MatImParwin),axis=0)


def writereport(namereport, freq, Tsample, user, totaldata, typedata,
Composition,model_pca):
    """
    Write JSON report into file
```

```
    :param namereport: name to save
    :param freq: freq tested
    :param Tsample: Tsamle tested
    :param user: user tested
    :param totaldata:total data windows tested
    :param typedata: type data train vq
    :return: write json file
    """

    Key = str(str(user)+str(freq)+str(Composition))
    report = {
        "Freqsample": freq,
        "Tsample": Tsample,
        "Composition": str(Composition),
        "User": user,
        "Totalwindows": totaldata,
        "WindowSize" : Tsample * freq,
        "TypeData" : str(typedata),
        "PCA explained_variance_ratio_2D"
:str(model_pca.explained_variance_ratio_),
        "PCA singular_values_2D" :str(model_pca.singular_values_)
    }

    try:
        jsonread = json.loads(open("JsonReport/"+namereport +
'.json').read())
        jsonread[Key] = report

        with open("JsonReport/"+namereport + ".json", "w") as f:

            json.dump(jsonread, f)
            f.close()
    except:
        INI = {}
        INI[Key] = report
        with open("JsonReport/"+namereport + ".json", "w") as f:
            json.dump(INI, f)
            f.close()


def PlotModelsVQ(x_tra, user, modelsMSCL, Composition, configjson, pcaplot):
    """
    Plot weight and data
    :param x_tra: Data 2d
    :param modelsMSCL: model objet MSCL
    :param Composition: number of axis sensor
    :param configjson: json config file
    :param pcaplot: model pca 2d train before
    :return: fig
    """
    date =
datetime.datetime.fromtimestamp(time.time()).strftime('%Y%m%d_%H%M%S')
    Data = pcaplot.transform(x_tra)
    nModels = len(searchJson(configjson, ['Compress_model', 'MSCL',
'TargetFuntion']))
    Neurons = searchJson(configjson, ['Compress_model', 'MSCL', 'Neurons'])
    cont = 1
    j = 0
    fig, axs = plt.subplots(nModels + 1, 1, sharex='all', sharey='all')
```

```
    fig.tight_layout()
    axs[0].scatter(Data[:, 0], Data[:, 1])
    for i in modelsMSCL:
        centroids = pcaplot.transform(modelsMSCL[i].weight)
        if (nModels >= cont):
            axs[cont].scatter(centroids[:, 0], centroids[:, 1])
            axs[cont].set_title(i)
        if (nModels == cont):
            cont = 0
            plt.savefig(
                "Plots/Trainings/MSCL/Compare_ModelsNeurons_" + str(user) +
str(Composition) + str(Neurons[j]) + str(
                    date) + '.png')
            fig, axs = plt.subplots(nModels + 1, 1, sharex='all',
sharey='all')
            fig.tight_layout()
            axs[0].scatter(Data[:, 0], Data[:, 1])
            j += 1

        cont += 1


def LabelCheck(Mat):
    """
    Check same label into a windows
    :param Mat: Mat type object
    :return:
    """
    LabelCheck = np.max(Mat, axis=1)
    LabelcheckMin = np.min(Mat, axis=1)

    comparate = (LabelCheck == LabelcheckMin)
    del (LabelcheckMin)
    search = np.where(comparate == False)
    del (comparate)
    LabelCheck[search] = "Notregistred"
    del (search)
    return np.reshape(LabelCheck, (len(LabelCheck), 1))


def trainPCAPLOT(MatWindow, configjson):
    """
    Train PCA scikit model to plot 2d
    :param MatWindow: Mat 2d
    :param configjson: config json file
    :return: PCA model fit
    """
    N_comp_plot = searchJson(configjson, ['Compress_model', 'PCA',
'N_comp_plot'])

    model_pca = decomposition.PCA(n_components=N_comp_plot)
    model_pca.fit(MatWindow)

    return model_pca


def trainPCA(MatWindow,n_comp):
    """
    Train PCA scikit model
```

```
    :param MatWindow: Mat 2d
    :param configjson: config json file
    :return:PCA model fit
    """

    model_pca = decomposition.PCA(n_components=n_comp)
    model_pca.fit(MatWindow)

    return model_pca

def trainVQ(x_tra,TF, configjson,Composition):
    """
    Train pymscl model
    :param x_tra:Train data array2d
    :param configjson: congif json file
    :param Composition: number of axis sensor
    :return:
    """

    Neurons = searchJson(configjson, ['Compress_model', 'MSCL', 'Neurons'])
    Batch = searchJson(configjson, ['Compress_model', 'MSCL', 'Batch'])
    Alphaini = searchJson(configjson, ['Compress_model', 'MSCL', 'Alphaini'])
    Alphaend = searchJson(configjson, ['Compress_model', 'MSCL', 'Alphaend'])
    Tepoch = searchJson(configjson, ['Compress_model', 'MSCL', 'Tepoch'])
    Trainsize = searchJson(configjson, ['Compress_model', 'MSCL',
'Train%data'])
    Validationsize = searchJson(configjson, ['Compress_model', 'MSCL',
'Validation%data'])
    Testsize = searchJson(configjson, ['Compress_model', 'MSCL',
'Test%data'])

    models = dict()
    x_tra, x_val = train_test_split(x_tra, test_size=Validationsize)


    for neuron in Neurons:
        name = ("TrainSize: " + str(Trainsize) +
                                "Composition: "+ str(Composition)+
                                " ,ValSize: " + str(Validationsize) +
                                " ,TestSize: " + str(Testsize) +
                                " ,TF: " + str(TF) +
                                " ,Neurons: " + str(neuron) +
                                " , alphaini: " + str(Alphaini) +
                                " , alphaend: " + str(Alphaend) +
                                " , Tepoch: " + str(Tepoch))
        print(name)
        model = pymscl.MSCL(neuron, Batch, Tepoch,Composition)
        model.fit_model(x_tra.astype(np.float16), x_val.astype(np.float16),
Alphaini, Alphaend, TF)
        #Save models
        models[str(TF)+str(neuron)] = model

    return models


def trainwindows(Mat, MatLabel, Composition, user, configjson):
    """
    * Select Ts = Sample_rate(Hz)
    * Take out the windows with Ts and T.
```

```
    * Calculate the means and the standard deviations.
    * Pass the data through the MSCL.
    :param Mat: Mat Scaled from Dataset
    :param MatLabel: Mat object from Dataset
    :param user: user train
    :param Composition: Composition components
    :configjson: config json
    """
    global CSV_save, contCSV
    Fsample = searchJson(configjson, ['PreProcess', 'Fs'])
    T = searchJson(configjson, ['PreProcess', 'T'])
    Stadistics = searchJson(configjson, ['PreProcess', 'Stadistics'])
    Ts_sensor = searchJson(configjson, ['DataBase', 'Ts_sensor'])
    TargetFuntion = searchJson(configjson, ['Compress_model', 'MSCL',
'TargetFuntion'])

    for period in T:
        for fs in Fsample:
            savename = ("_User_"+str(user)+",Comp_ " + str(Composition) +
",T_" + str(period) + ",Ts_"+ str(fs))
            WindowSize = int( period * fs )
            # Submuestro
            if (fs != Ts_sensor):  # Tengo que submuestrear

                MatWindow = Submuestreo(Mat, Composition, WindowSize, fs,
Ts_sensor)
                MatLabelWindow = Submuestreo(MatLabel, Composition,
WindowSize, fs, Ts_sensor)

            else:  # Freq sensor

                MatWindow = Windows_Maker(Mat, Composition, WindowSize)
                MatLabelWindow = Windows_Maker(MatLabel, Composition,
WindowSize)

            # Stadistics
            MatWindow ,MatStadistics = Apply_Stadistics(MatWindow,
Stadistics, savename)
            #Labels
            MatLabelWindow = LabelCheck(MatLabelWindow)

            #Training_Test Data
            Testsize = searchJson(configjson, ['Compress_model', 'MSCL',
'Test%data'])
            x_tra, x_test = train_test_split(MatWindow, test_size=Testsize)
            lb_tra, lb_test = train_test_split(MatLabelWindow,
test_size=Testsize)
            #savemats
            try:
                np.save('Trains/Mats/'
                        +str(user)+ str(Composition) + str(period) +
str(fs)  + "x_tra.npy", x_tra)
                np.save('Trains/Mats/'
                        +str(user)+ str(Composition) + str(period) +
str(fs)  + "x_test.npy", x_test)
                np.save('Trains/Mats/'
                        +str(user)+ str(Composition) + str(period) +
str(fs)  + "lb_tra.npy", lb_tra)
                np.save('Trains/Mats/'
```

```
                            +str(user)+ str(Composition) + str(period) +
str(fs)  + "lb_test.npy", lb_test)
            except:
                print("Fail saving x_tra,x_test,lb_tra,lb_test")

            #Train compress model
            x_tra_permutation = np.random.permutation(x_tra)
            pcaplot = trainPCAPLOT(x_tra_permutation, configjson) #PCA en 2d
para plotear
            datorealtra = (x_tra *
np.reshape(MatStadistics[0:x_tra.shape[0],1],(x_tra.shape[0],1))) +
np.reshape(MatStadistics[0:x_tra.shape[0],0],(x_tra.shape[0],1))
            datorealtest = (x_test *
np.reshape(MatStadistics[0:x_test.shape[0],1],(x_test.shape[0],1))) +
np.reshape(MatStadistics[0:x_test.shape[0],0],(x_test.shape[0],1))
            N_comp_evaluate = searchJson(configjson, ['Compress_model',
'PCA', 'N_comp_evaluate'])
            Pcamsetra = list()
            Pcamsetest = list()
            for component in range(N_comp_evaluate):
                pcamodel = trainPCA(x_tra_permutation, component)
                xpre_tra =
pcamodel.inverse_transform(pcamodel.transform(x_tra)).astype(np.float16)
                xpre_test =
pcamodel.inverse_transform(pcamodel.transform(x_test)).astype(np.float16)

                datopretra =  (xpre_tra  *
np.reshape(MatStadistics[0:xpre_tra.shape[0],1],(xpre_tra.shape[0],1))) +
np.reshape(MatStadistics[0:xpre_tra.shape[0],0],(xpre_tra.shape[0],1))
                datopretest = (xpre_test *
np.reshape(MatStadistics[0:xpre_test.shape[0],1],(xpre_test.shape[0],1))) +
np.reshape(MatStadistics[0:xpre_test.shape[0],0],(xpre_test.shape[0],1))
                msetra = np.mean((datorealtra - datopretra) ** 2,
axis=1).astype(np.float16)
                msetest = np.mean((datorealtest - datopretest) ** 2,
axis=1).astype(np.float16)

                Pcamsetra.append(np.mean(msetra))
                Pcamsetest.append(np.mean(msetest))
                try:
                    pickle.dump(pcamodel, open(
                        "Trains/MatsPCA/pcamodel_" + str(user) + "_" +
str(Composition) + "_" + str(fs) + "_" + str(
                            period) + "_" + str(component) + ".p", "wb"))
                    pickle.dump(pcaplot, open(
                        "Trains/MatsPCA/pcaplot_" + str(user) + "_" +
str(Composition) + "_" + str(fs) + "_" + str(
                            period) + ".p", "wb"))
                except:
                    print("Fail savepca")

            #Model TF
            for TF in TargetFuntion:
                modelsVQ =
trainVQ(x_tra_permutation,TF,configjson,Composition) #VQMODEL con parametros
json

                #savemodels.
                try:
```

```
                    pickle.dump(modelsVQ, open("Trains/MatsVQ/modelsVQ_"
+str(user)+"_"+str(Composition)+"_"+str(TF)+"_"+str(fs)+"_"+str(period)+".p",
"wb"))
                except:
                    print("Fail savemodels")

                for model in modelsVQ:
                    xpre_tra, querrors_tra, Neuronwins_tra =
modelsVQ[model].predict(x_tra)
                    xpre_test, querrors_test, Neuronwins_test =
modelsVQ[model].predict(x_test)

                    datopretra   = (xpre_tra  *
np.reshape(MatStadistics[0:xpre_tra.shape[0],1],(xpre_tra.shape[0],1))) +
np.reshape(MatStadistics[0:xpre_tra.shape[0],0],(xpre_tra.shape[0],1))
                    datopretest  = (xpre_test *
np.reshape(MatStadistics[0:xpre_test.shape[0],1],(xpre_test.shape[0],1))) +
np.reshape(MatStadistics[0:xpre_test.shape[0],0],(xpre_test.shape[0],1))
                    msetra  = np.mean((datorealtra - datopretra) ** 2,
axis=1).astype(np.float16)
                    msetest = np.mean((datorealtest - datopretest) ** 2,
axis=1).astype(np.float16)
                    contCSV += 1
                    CSV_save.loc[contCSV] = [user, fs,
period,Composition,model,modelsVQ[model].neurons,modelsVQ[model].weight.size,
(math.log(modelsVQ[model].neurons)/math.log(2)),NBSTD,np.mean(msetra),np.mean
(msetest),Pcamsetra,Pcamsetest]

                #PlotDatamodelsMSCL

PlotModelsVQ(x_tra,user,modelsVQ,Composition,configjson,pcaplot)

PlotPSNR(datorealtra,x_tra,MatStadistics,TF,modelsVQ,period,fs,user,configjso
n)

         # Del data
        del (MatWindow)
        del (MatLabelWindow)

def
PlotPSNR(datareal,datastaditic,staditic,TarFunct,modelsvq,Ts,fs,user,configjs
on):
    """
    Box plot PSNR
    :param datareal: real data without stadistics
    :param datastaditic: data with stadistics process
    :param staditic: Mat stadistics
    :param TarFunct: Target function
    :param modelsvq: pymscl object
    :param Ts: Time to save window
    :param fs: Frec sample
    :param user: user train
    :param configjson: json file
    :return: fig
    """
    sensorRange = searchJson(configjson, ['DataBase', 'SensorRange'])
    N_comp_evaluate = searchJson(configjson, ['Compress_model','PCA',
'N_comp_evaluate'])
    date =
```

```
datetime.datetime.fromtimestamp(time.time()).strftime('%Y%m%d_%H%M%S')
    figPCAPSNR, axPCA = plt.subplots(figsize=(10, 10))
    axPCA.set_title('PSNR PCA')
    axPCA.set_ylabel('PSNR')
    #PCAMODEL
    PSNRPCA =list()
    PCAlabel =list()
    for ncom in range(N_comp_evaluate):

        pcamodel = trainPCA(datastaditic,ncom)
        xpre_tra =
pcamodel.inverse_transform(pcamodel.transform(datastaditic)).astype(np.float1
6)
        xpre_tra = (xpre_tra  *
np.reshape(staditic[0:xpre_tra.shape[0],1],(xpre_tra.shape[0],1))) +
np.reshape(staditic[0:xpre_tra.shape[0],0],(xpre_tra.shape[0],1)).astype(np.f
loat16)
        mse = np.mean( (datareal - xpre_tra) ** 2 ,axis=1).astype(np.float16)
        for i in range(mse.shape[0]):
            PSNR = mse
            PSNR[i] = 10 *
math.log10(sensorRange*sensorRange/(mse[i]+EPSILON))

        PSNRPCA.append((PSNR))
        PCAlabel.append(("PCA"+str(ncom)+"("+str(ncom*16)+")"))
    axPCA.set_xticklabels(labels =PCAlabel,rotation=45, fontsize=8)
    axPCA.boxplot(PSNRPCA)

plt.savefig("Plots/Trainings/PSNR/PCAPSNR"+str(user)+"_T_"+str(Ts)+"_Fs_"+str
(fs)+"_"+str(date)+".png")

    figVQPSNR, axVQ = plt.subplots(figsize=(10, 10))
    axVQ.set_title('PSNRModelVQ'+str(TarFunct))
    axVQ.set_ylabel('PSNR')
    axVQ.set_xlabel('Neurons (bitsenvio)')
    PSNRVQ = list()
    VQlabel = list()
    for nvq  in modelsvq:

        xpre_tra, querrors_tra, Neuronwins_tra =
modelsvq[nvq].predict(datastaditic)
        xpre_tra = (xpre_tra  *
np.reshape(staditic[0:xpre_tra.shape[0],1],(xpre_tra.shape[0],1))) +
np.reshape(staditic[0:xpre_tra.shape[0],0],(xpre_tra.shape[0],1)).astype(np.f
loat16)
        mse = np.mean((datareal - xpre_tra) ** 2, axis=1).astype(np.float16)
        for i in range(mse.shape[0]):
            PSNR = mse
            PSNR[i] = 10 *
math.log10(sensorRange*sensorRange/(mse[i]+EPSILON))
        PSNRVQ.append((PSNR))

VQlabel.append((str(modelsvq[nvq].neurons)+"("+str((math.log(modelsvq[nvq].ne
urons)/math.log(2)))+")"))
    axVQ.set_xticklabels(labels=VQlabel,rotation=45, fontsize=8)
    axVQ.boxplot(PSNRVQ)


plt.savefig("Plots/Trainings/PSNR/VQPSNR"+str(user)+str(TarFunct)+"_T_"+str(T
```

```
s)+"_Fs_"+str(fs)+"_"+str(date)+".png")


"""
=====================================================
MAIN
=====================================================
Autor: Jorge Vizarraga Turmo

Date: 30/10/2019

Version:1.0

Flowchart
-----------------------------------------------------

* Test Users vs activities

"""

# Start Logg
logging.basicConfig(format='%(asctime)s - %(message)s', level=logging.INFO)


usertrain = searchJson(configjson, ['DataBase', 'UserTrain'])
CSV_save =
pd.DataFrame(columns=["User","fs","period","Composition","model","Neuronas","
SizeVQ","BitsEnvioVQ","BitsStaditics","MSE_meanVQ_train","MSE_meanVQ_test","M
SE_meanPCAtra1-10comp","MSE_meanPCAtest1-10comp"])
contCSV = 0

# Load database for userTrain
MatTrain, MatLabels = MatDatabase(configjson, usertrain)
# StartTrainPCA
trainDataVQ(MatTrain, MatLabels, usertrain, configjson)

timesave=datetime.datetime.fromtimestamp(time.time()).strftime('%Y%m%d_%H%M%')
CSV_save.to_csv("JsonReport/CSVTrain_"+str(usertrain)+str(timesave)+".csv")
```

2.4. Pruebas MSCL FULL

User	fs	period	Composition	model	Neuronas	SizeVQ	BitsEnvioVQ	BitsStd	MSE_meanVQ_train	MSE_meanVQ_test
d	25	0.6	Full	ESCL1024	1024	46080	10.0	32	0.57373046875	0.560546875
d	25	0.6	Full	ESCL2048	2048	92160	11.0	32	0.383544921875	0.423828125
d	25	0.6	Full	ESCL4096	4096	184320	12.0	32	0.2236328125	0.3173828125
d	25	0.6	Full	ESCL8192	8192	368640	13.0	32	0.119873046875	0.234619140625
d	50	0.6	Full	ESCL1024	1024	92160	10.0	32	0.51611328125	0.52392578125
d	50	0.6	Full	ESCL2048	2048	184320	11.0	32	0.3330078125	0.44189453125
d	50	0.6	Full	ESCL4096	4096	368640	12.0	32	0.1759033203125	0.345703125
d	50	0.6	Full	ESCL8192	8192	737280	13.0	32	0.08782958984375	0.28369140625
d	100	0.6	Full	ESCL1024	1024	184320	10.0	32	0.47412109375	0.80322265625
d	100	0.6	Full	ESCL2048	2048	368640	11.0	32	0.317626953125	0.73388671875
d	100	0.6	Full	ESCL4096	4096	737280	12.0	32	0.1783447265625	0.7041015625
d	100	0.6	Full	ESCL8192	8192	1474560	13.0	32	0.1361083984375	0.6982421875

MSE_meanPCAtra1-10comp	MSE_meanPCAtest1-10comp
[3.742, 2.137, 1.752, 1.416, 1.2705, 1.141, 1.035, 0.944, 0.862, 0.788]	[3.45, 1.896, 1.565, 1.251, 1.121, 0.9985, 0.915, 0.843, 0.764, 0.707]
[3.742, 2.137, 1.752, 1.416, 1.2705, 1.141, 1.035, 0.944, 0.862, 0.788]	[3.45, 1.896, 1.565, 1.251, 1.121, 0.9985, 0.915, 0.843, 0.764, 0.707]
[3.742, 2.137, 1.752, 1.416, 1.2705, 1.141, 1.035, 0.944, 0.862, 0.788]	[3.45, 1.896, 1.565, 1.251, 1.121, 0.9985, 0.915, 0.843, 0.764, 0.707]

User	fs	period	Composition	model	Neuronas	SizeVQ	BitsEnvioVQ	BitsStd	MSE_meanVQ_train	MSE_meanVQ_test
e	25	1.6	Full	unity1024	1024	122880	10.0	32	1.16796875	1.50390625
e	25	1.6	Full	unity2048	2048	245760	11.0	32	0.75390625	1.1064453125
e	25	1.6	Full	unity4096	4096	491520	12.0	32	0.42041015625	0.81640625
e	25	1.6	Full	ESCL1024	1024	122880	10.0	32	1.111328125	1.458984375
e	25	1.6	Full	ESCL2048	2048	245760	11.0	32	0.82421875	1.169921875
e	25	1.6	Full	ESCL4096	4096	491520	12.0	32	0.428955078125	0.80810546875
e	25	1.6	Full	FSCL1024	1024	122880	10.0	32	1.107421875	1.4013671875
e	25	1.6	Full	FSCL2048	2048	245760	11.0	32	0.75927734375	1.1376953125
e	25	1.6	Full	FSCL4096	4096	491520	12.0	32	0.4150390625	0.8232421875
e	50	1.6	Full	unity1024	1024	245760	10.0	32	1.0312521205	1.3603515625
e	50	1.6	Full	unity2048	2048	491520	11.0	32	0.59033203125	1.0703125
e	50	1.6	Full	unity4096	4096	983040	12.0	32	0.334228515625	0.86376953125
e	50	1.6	Full	ESCL1024	1024	245760	10.0	32	1.017578125	1.4052734375
e	50	1.6	Full	ESCL2048	2048	491520	11.0	32	0.69482421875	1.1474609375
e	50	1.6	Full	ESCL4096	4096	983040	12.0	32	0.3603515625	0.83935546875
e	50	1.6	Full	FSCL1024	1024	245760	10.0	32	1.125	1.4755859375
e	50	1.6	Full	FSCL2048	2048	491520	11.0	32	0.640625	1.0498046875
e	50	1.6	Full	FSCL4096	4096	983040	12.0	32	0.300048828125	0.7451171875
e	100	1.6	Full	unity1024	1024	491520	10.0	32	0.99462890625	1.392578125
e	100	1.6	Full	unity2048	2048	983040	11.0	32	0.5849609375	1.3837890625
e	100	1.6	Full	unity4096	4096	1966080	12.0	32	0.406494140625	1.3603515625
e	100	1.6	Full	ESCL1024	1024	491520	10.0	32	1.078125	1.498046875
e	100	1.6	Full	ESCL2048	2048	983040	11.0	32	0.70751953125	1.390625
e	100	1.6	Full	ESCL4096	4096	1966080	12.0	32	0.424072265625	1.357421875
e	100	1.6	Full	FSCL1024	1024	491520	10.0	32	1.0234375	1.48046875
e	100	1.6	Full	FSCL2048	2048	983040	11.0	32	0.6181640625	1.4072265625

e	100	1.6	Full	FSCL4096	4096	1966080	12.0	32	0.37744140625	1.3720703125

fs	T	MSE_meanPCAtra1-10comp	MSE_meanPCAtest1-10comp
25	1.6	[4.77, 3.326, 2.947, 2.61, 2.416, 2.244, 2.084, 1.94, 1.83, 1.7295]	[4.926, 3.486, 3.158, 2.822, 2.613, 2.418, 2.244, 2.105, 1.989, 1.882]
50	1.6	[4.66, 3.238, 2.883, 2.566, 2.379, 2.21, 2.057, 1.919, 1.813, 1.713]	[4.656, 3.387, 3.012, 2.623, 2.39, 2.215, 2.049, 1.909, 1.812, 1.714]
100	1.6	[4.816, 3.365, 2.996, 2.656, 2.455, 2.273, 2.111, 1.97, 1.854, 1.754]	[3.82, 2.652, 2.375, 2.12, 1.921, 1.803, 1.685, 1.582, 1.511, 1.431]

2.5. Pruebas MSCL Single

User	fs	period	Composition	model	Neuronas	SizeVQ	BitsEnvioVQ	BitsStd	MSE_meanVQ_train	MSE_meanVQ_test
d	25	1	Single0	ESCL1024	1024	25600	10.0	32	0.666015625	0.630859375
d	25	1	Single0	ESCL2048	2048	51200	11.0	32	0.466552734375	0.55419921875
d	25	1	Single0	ESCL4096	4096	102400	12.0	32	0.281494140625	0.461669921875
d	50	1	Single0	ESCL1024	1024	51200	10.0	32	0.625	0.014495849609375
d	50	1	Single0	ESCL2048	2048	102400	11.0	32	0.41796875	0.0166778564453125
d	50	1	Single0	ESCL4096	4096	204800	12.0	32	0.220703125	0.01458740234375
d	100	1	Single0	ESCL1024	1024	102400	10.0	32	0.60791015625	0.0016994476318359375
d	100	1	Single0	ESCL2048	2048	204800	11.0	32	0.33837890625	0.0016660690307617188
d	100	1	Single0	ESCL4096	4096	409600	12.0	32	0.242919921875	0.0016565322875976562
d	25	1.6	Single0	ESCL1024	1024	40960	10.0	32	0.67041015625	0.72998046875
d	25	1.6	Single0	ESCL2048	2048	81920	11.0	32	0.369140625	0.5927734375
d	25	1.6	Single0	ESCL4096	4096	163840	12.0	32	0.1925048828125	0.50048828125
d	50	1.6	Single0	ESCL1024	1024	81920	10.0	32	0.55322265625	0.004974365234375
d	50	1.6	Single0	ESCL2048	2048	163840	11.0	32	0.322265625	0.0044403076171875
d	50	1.6	Single0	ESCL4096	4096	327680	12.0	32	0.1890869140625	0.004364013671875
d	100	1.6	Single0	ESCL1024	1024	163840	10.0	32	0.5234375	0.0019235610961914062
d	100	1.6	Single0	ESCL2048	2048	327680	11.0	32	0.349853515625	0.0019159317016601562
d	100	1.6	Single0	ESCL4096	4096	655360	12.0	32	0.31396484375	0.0019168853759765625
d	25	1	Single1	ESCL1024	1024	25600	10.0	32	0.59716796875	0.41015625
d	25	1	Single1	ESCL2048	2048	51200	11.0	32	0.40869140625	0.350341796875
d	25	1	Single1	ESCL4096	4096	102400	12.0	32	0.2427978515625	0.304931640625
d	50	1	Single1	ESCL1024	1024	51200	10.0	32	0.53076171875	0.0008425712585449219
d	50	1	Single1	ESCL2048	2048	102400	11.0	32	0.343017578125	0.0007534027099609375
d	50	1	Single1	ESCL4096	4096	204800	12.0	32	0.20068359375	0.0007119178771972656

d	100	1	Single1	ESCL1024	1024	102400	10.0	32	0.5751953125	0.003948211669921875
d	100	1	Single1	ESCL2048	2048	204800	11.0	32	0.349609375	0.00390625
d	100	1	Single1	ESCL4096	4096	409600	12.0	32	0.24072265625	0.003887176513671875
d	25	1.6	Single1	ESCL1024	1024	40960	10.0	32	0.6025390625	0.470703125
d	25	1.6	Single1	ESCL2048	2048	81920	11.0	32	0.4033203125	0.391357421875
d	25	1.6	Single1	ESCL4096	4096	163840	12.0	32	0.195068359375	0.32470703125
d	50	1.6	Single1	ESCL1024	1024	81920	10.0	32	0.4765625	0.00159454345703125
d	50	1.6	Single1	ESCL2048	2048	163840	11.0	32	0.2490234375	0.0014848709106445312
d	50	1.6	Single1	ESCL4096	4096	327680	12.0	32	0.146240234375	0.0014467239379882812
d	100	1.6	Single1	ESCL1024	1024	163840	10.0	32	0.51904296875	0.00484466552734375
d	100	1.6	Single1	ESCL2048	2048	327680	11.0	32	0.33544921875	0.0048675537109375
d	100	1.6	Single1	ESCL4096	4096	655360	12.0	32	0.293212890625	0.0048675537109375
d	25	1	Single2	ESCL1024	1024	25600	10.0	32	##########	0.90087890625
d	25	1	Single2	ESCL2048	2048	51200	11.0	32	0.82177734375	0.75244140625
d	25	1	Single2	ESCL4096	4096	102400	12.0	32	0.48681640625	0.65966796875
d	50	1	Single2	ESCL1024	1024	51200	10.0	32	##########	0.0027523040771484375
d	50	1	Single2	ESCL2048	2048	102400	11.0	32	0.67919921875	0.002593994140625
d	50	1	Single2	ESCL4096	4096	204800	12.0	32	0.396728515625	0.0023784637451171875
d	100	1	Single2	ESCL1024	1024	102400	10.0	32	##########	0.003040313720703125
d	100	1	Single2	ESCL2048	2048	204800	11.0	32	0.6865234375	0.0028667449951171875
d	100	1	Single2	ESCL4096	4096	409600	12.0	32	0.505859375	0.002857208251953125
d	25	1.6	Single2	ESCL1024	1024	40960	10.0	32	##########	##########
d	25	1.6	Single2	ESCL2048	2048	81920	11.0	32	0.70703125	0.7978515625
d	25	1.6	Single2	ESCL4096	4096	163840	12.0	32	0.384033203125	0.64599609375
d	50	1.6	Single2	ESCL1024	1024	81920	10.0	32	0.9560546875	0.003021240234375
d	50	1.6	Single2	ESCL2048	2048	163840	11.0	32	0.50732421875	0.002552032470703125
d	50	1.6	Single2	ESCL4096	4096	327680	12.0	32	0.327392578125	0.0022373199462890625

d	100	1.6	Single2	ESCL1024	1024	163840	10.0	32	##########	0.003459930419921875
d	100	1.6	Single2	ESCL2048	2048	327680	11.0	32	0.64892578125	0.0034084320068359375
d	100	1.6	Single2	ESCL4096	4096	655360	12.0	32	0.58984375	0.0034084320068359375

MSE_meanPCAtra1-10comp	MSE_meanPCAtest1-10comp
[2.19, 1.797, 1.417, 1.238, 1.115, 1.004, 0.9146, 0.8267, 0.743, 0.679]	[1.674, 1.36, 1.093, 0.9624, 0.8647, 0.777, 0.7104, 0.6484, 0.579, 0.534]
[2.256, 1.849, 1.452, 1.277, 1.156, 1.046, 0.9585, 0.878, 0.7974, 0.7373]	[0.0258, 0.02104, 0.0202, 0.01482, 0.01462, 0.01433, 0.01221, 0.01075, 0.01057, 0.0103]
[2.545, 2.068, 1.634, 1.456, 1.314, 1.193, 1.094, 0.9985, 0.9077, 0.8447]	[0.00284, 0.00247, 0.001976, 0.001774, 0.001661, 0.001461, 0.001401, 0.001279, 0.001167, 0.001117]
[2.219, 1.902, 1.58, 1.374, 1.267, 1.163, 1.072, 1.015, 0.952, 0.896]	[1.695, 1.485, 1.262, 1.106, 1.026, 0.931, 0.8516, 0.8027, 0.761, 0.7173]
[2.283, 1.947, 1.652, 1.426, 1.317, 1.209, 1.116, 1.056, 0.9966, 0.9424]	[0.03162, 0.03052, 0.01492, 0.01435, 0.01355, 0.0133, 0.01292, 0.01225, 0.01177, 0.011734]
[2.56, 2.182, 1.805, 1.563, 1.445, 1.333, 1.237, 1.156, 1.086, 1.02]	[0.004414, 0.003906, 0.002502, 0.002228, 0.002117, 0.002003, 0.001893, 0.001843, 0.0017605, 0.001565]
[1.653, 1.476, 1.31, 1.17, 1.029, 0.9316, 0.834, 0.748, 0.666, 0.5874]	[0.9414, 0.824, 0.7354, 0.658, 0.571, 0.516, 0.4685, 0.4207, 0.3772, 0.3293]
[1.611, 1.441, 1.287, 1.154, 1.033, 0.945, 0.85, 0.767, 0.694, 0.6245]	[0.002703, 0.002537, 0.002365, 0.002052, 0.001889, 0.001788, 0.001578, 0.0014105, 0.001347, 0.001136]
[1.868, 1.681, 1.511, 1.359, 1.206, 1.094, 1.004, 0.908, 0.823, 0.7427]	[0.00434, 0.004086, 0.003963, 0.003868, 0.00375, 0.003298, 0.003141, 0.002827, 0.002726, 0.002495]
[1.662, 1.515, 1.384, 1.283, 1.177, 1.106, 1.037, 0.9756, 0.918, 0.8623]	[0.946, 0.862, 0.7656, 0.7026, 0.6587, 0.619, 0.5815, 0.547, 0.5205, 0.4873]
[1.616, 1.479, 1.351, 1.25, 1.151, 1.085, 1.022, 0.96, 0.904, 0.848]	[0.003723, 0.003588, 0.003405, 0.002531, 0.00234, 0.002062, 0.001991, 0.001957, 0.001923, 0.001877]

[1.876, 1.708, 1.571, 1.439, 1.333, 1.258, 1.187, 1.112, 1.049, 0.9907]	[0.006145, 0.006004, 0.005886, 0.00499, 0.00424, 0.004185, 0.00399, 0.003756, 0.003716, 0.003386]
[3.596, 3.06, 2.541, 2.283, 2.031, 1.84, 1.649, 1.471, 1.292, 1.119]	[2.402, 2.008, 1.626, 1.454, 1.317, 1.191, 1.069, 0.938, 0.8306, 0.7144]
[3.658, 3.123, 2.63, 2.367, 2.143, 1.951, 1.762, 1.581, 1.401, 1.225]	[0.006046, 0.005424, 0.004383, 0.003918, 0.003286, 0.003057, 0.002773, 0.002571, 0.00235, 0.002045]
[4.13, 3.47, 2.885, 2.604, 2.348, 2.146, 1.948, 1.766, 1.585, 1.395]	[0.00535, 0.004723, 0.003792, 0.0035, 0.003183, 0.002777, 0.002481, 0.002256, 0.001922, 0.001666]
[3.62, 3.127, 2.713, 2.531, 2.36, 2.191, 2.043, 1.911, 1.7705, 1.647]	[2.402, 2.072, 1.769, 1.646, 1.552, 1.437, 1.337, 1.246, 1.162, 1.084]
[3.672, 3.193, 2.764, 2.57, 2.398, 2.238, 2.086, 1.956, 1.816, 1.699]	[0.00605, 0.005295, 0.004772, 0.00457, 0.004395, 0.00423, 0.003983, 0.003822, 0.003647, 0.002996]
[4.145, 3.639, 3.113, 2.904, 2.717, 2.525, 2.35, 2.203, 2.037, 1.901]	[0.005486, 0.004337, 0.00391, 0.0037, 0.003422, 0.003214, 0.002964, 0.002848, 0.002523, 0.00234]

User	fs	period	Composition	model	Neuronas	SizeVQ	BitsEnvioVQ	BitsStaditics	MSE_meanVQ_train	MSE_meanVQ_test
e	25	1	Single0	ESCL1024	1024	25600	10.0	32	1.28515625	1.28515625
e	25	1	Single0	ESCL2048	2048	51200	11.0	32	1.0322265625	1.0322265625
e	25	1	Single0	ESCL4096	4096	102400	12.0	32	0.72412109375	0.97412109375
e	50	1	Single0	ESCL1024	1024	51200	10.0	32	1.1884765625	1.041015625
e	50	1	Single0	ESCL2048	2048	102400	11.0	32	0.84619140625	0.9208984375
e	50	1	Single0	ESCL4096	4096	204800	12.0	32	0.56298828125	0.779296875
e	100	1	Single0	ESCL1024	1024	102400	10.0	32	1.1494140625	0.00720596313476562
e	100	1	Single0	ESCL2048	2048	204800	11.0	32	0.86865234375	0.00715255737304687
e	100	1	Single0	ESCL4096	4096	409600	12.0	32	0.52099609375	0.00704574584960937
e	25	1	Single1	ESCL1024	1024	25600	10.0	32	1.17578125	1.0927734375
e	25	1	Single1	ESCL2048	2048	51200	11.0	32	0.931640625	0.92724609375
e	25	1	Single1	ESCL4096	4096	102400	12.0	32	0.6533203125	0.80419921875
e	50	1	Single1	ESCL1024	1024	51200	10.0	32	1.00390625	0.85302734375
e	50	1	Single1	ESCL2048	2048	102400	11.0	32	0.73388671875	0.7841796875

e	50	1	Single1	ESCL4096	4096	204800	12.0	32	0.482421875	0.67822265625
e	100	1	Single1	ESCL1024	1024	102400	10.0	32	0.99951171875	0.003217697143554687 5
e	100	1	Single1	ESCL2048	2048	204800	11.0	32	0.673828125	0.002979278564453125
e	100	1	Single1	ESCL4096	4096	409600	12.0	32	0.4091796875	0.002988815307617187 5
e	25	1.6	Single2	FSCL1024	1024	40960	10.0	32	2.796875	2.685546875
e	25	1.6	Single2	FSCL2048	2048	81920	11.0	32	1.994140625	2.37890625
e	25	1.6	Single2	FSCL4096	4096	163840	12.0	32	1.30078125	1.962890625
e	50	1.6	Single2	FSCL1024	1024	81920	10.0	32	2.228515625	1.9892578125
e	50	1.6	Single2	FSCL2048	2048	163840	11.0	32	1.435546875	1.8212890625
e	50	1.6	Single2	FSCL4096	4096	327680	12.0	32	0.78955078125	1.548828125
e	100	1.6	Single2	FSCL1024	1024	163840	10.0	32	2.138671875	0.0088958740234375
e	100	1.6	Single2	FSCL2048	2048	327680	11.0	32	1.3056640625	0.00872039794921875
e	100	1.6	Single2	FSCL4096	4096	655360	12.0	32	0.82080078125	0.00867462158203125

Comp	Fs	MSE_meanPCAtra1-10comp	MSE_meanPCAtest1-10comp
Single0	25	[3.102, 2.633, 2.182, 1.989, 1.822, 1.668, 1.526, 1.389, 1.263, 1.14]	[2.717, 2.352, 1.98, 1.787, 1.646, 1.503, 1.372, 1.242, 1.129, 1.015]
Single0	50	[2.791, 2.375, 1.961, 1.801, 1.654, 1.525, 1.413, 1.296, 1.191, 1.085]	[2.023, 1.728, 1.408, 1.286, 1.178, 1.081, 0.9893, 0.9077, 0.829, 0.762]
Single0	100	[2.969, 2.531, 2.107, 1.932, 1.774, 1.638, 1.503, 1.382, 1.267, 1.16]	[0.011116, 0.01003, 0.009026, 0.00872, 0.007767, 0.007404, 0.007156, 0.00547, 0.005013, 0.004673]
Single1	25	[2.93, 2.48, 2.07, 1.823, 1.582, 1.43, 1.284, 1.141, 1.015, 0.887]	[2.44, 2.07, 1.727, 1.513, 1.314, 1.202, 1.079, 0.9614, 0.846, 0.744]
Single1	50	[2.635, 2.232, 1.858, 1.636, 1.428, 1.298, 1.176, 1.06, 0.9487, 0.843]	[1.772, 1.514, 1.28, 1.134, 0.998, 0.9146, 0.837, 0.749, 0.6763, 0.6016]
Single1	100	[2.71, 2.305, 1.917, 1.692, 1.476, 1.345, 1.222, 1.102, 0.9863, 0.874]	[0.00516, 0.004604, 0.003633, 0.003325, 0.003027, 0.002836, 0.002651, 0.002413, 0.002249, 0.00207]
Single2	25	[6.31, 5.438, 4.65, 4.34, 4.043, 3.762, 3.516, 3.273, 3.059, 2.848]	[5.312, 4.504, 3.83, 3.54, 3.307, 3.09, 2.904, 2.729, 2.53, 2.37]
Single2	50	[5.64, 4.906, 4.223, 3.934, 3.648, 3.4, 3.18, 2.973, 2.787, 2.582]	[4.023, 3.467, 2.9, 2.705, 2.527, 2.34, 2.184, 2.068, 1.933, 1.817]
Single2	100	[5.94, 5.098, 4.402, 4.113, 3.836, 3.566, 3.34, 3.133, 2.941, 2.748]	[0.01184, 0.01009, 0.00913, 0.00854, 0.00807, 0.00733, 0.007072, 0.006706, 0.006508, 0.006203]

3. Modelo clasificador

Script generación tabla

```
"""
------------------------------------------------------

======================================================
Classification Test
======================================================
Autor: Jorge Vizarraga Turmo

Version:3.2

------------------------------------------------------

"""

import numpy as np
from os import listdir
import pickle
import datetime
import pandas as pd
import time
import json
import logging
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
import pymscl
from sklearn import decomposition
from itertools import cycle
import math
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

from mpl_toolkits.mplot3d import Axes3D


def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    """
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    ---------
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from
matplotlib.pyplot.cm
                  see
http://matplotlib.org/examples/color/colormaps_reference.html
                  plt.get_cmap('jet') or plt.cm.Blues
```

```
    normalize:    If False, plot the raw numbers
                  If True, plot the proportions

    Usage
    -----
    plot_confusion_matrix(cm           = cm,                  # confusion
matrix created by
                                                              #
sklearn.metrics.confusion_matrix
                          normalize    = True,                # show
proportions
                          target_names = y_labels_vals,       # list of names
of the classes
                          title        = best_estimator_name) # title of
graph

    Citiation
    ---------
    http://scikit-
learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / np.sum(cm).astype('float')
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Greens')

    plt.figure(figsize=(8, 8))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

    date =
```

```
datetime.datetime.fromtimestamp(time.time()).strftime('%Y%m%d_%H%M%S')
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f};
misclass={:0.4f}'.format(accuracy, misclass))
    plt.savefig("Plots/Trainings/Tabla/Confumatrix"+str(date)+"png")
    plt.show()

def searchJson(file, search):
    """
    Function to search into JSON giving a file.json and the search sequence
    :param file: path with the name of the file
    :param search: search sequence in list [search1,search2,...,searchn]
    :return: search result
    """
    try:
        jsonread = json.loads(open(file + '.json').read())
    except Exception as e:
        logging.error("Exception occurred", exc_info=True)
        result = ""
    if (len(search) < 2):
        result = jsonread.get(search[0])
    else:
        result = jsonread.get(search[0])

    for i in range(len(search) - 1):
        result = result.get(search[i + 1])

    return result


def PlotReconstruccionError(x_tra,x_pre,querrorwinners,model):
    """
    Plot real data and predict data
    :param x_tra: Mat 2d real data
    :param x_pre: Mat 2d predict data
    :param querrorwinners: array 2d winners neurons
    :param model: pymscl object
    :return:
    """
    fig, axs = plt.subplots(4, 1)
    fig.tight_layout()
    date =
datetime.datetime.fromtimestamp(time.time()).strftime('%Y%m%d_%H%M%S')
    axs[0].scatter(x_tra[:, 0], x_tra[:, 1],c=np.random.rand(3,))
    axs[0].set_title("Realdata")
    axs[1].scatter(x_pre[:, 0], x_pre[:, 1],c=np.random.rand(3,))
    axs[1].set_title("Predictdata")
    axs[2].scatter(x_tra [:, 0], x_tra[:, 1],c=np.random.rand(3,))
    axs[2].scatter(x_pre[:, 0], x_pre[:, 1],c=np.random.rand(3,))
    axs[2].set_title("Realdata&Predictdata")
    axs[3].plot(np.array(range(querrorwinners.shape[0] )),
querrorwinners,c=np.random.rand(3,))
    axs[3].set_title("Qerror")

plt.savefig("Plots/Trainings/Predict/Predict"+str(model)+str(date)+".png")


def test(configjson):
```

```
    """
    Make test and draw prediction and confusion mats
    :param configjson: json config file
    :return:
    """
    global TableTrain,predictions,x_test,lb_test,lb_tra,x_tra,vqmodel

    path = searchJson(configjson, ['Predict', 'path'])
    vqmodel = pickle.load(open(path+"modelsVQ.p", 'rb'))
    pcaplot = pickle.load(open(path+"pcaplot.p", 'rb'))
    x_tra = np.load(path+"x_tra.npy")
    x_test = np.load(path + "x_test.npy")
    lb_tra = np.load(path + "lb_tra.npy",allow_pickle=True)
    lb_test = np.load(path + "lb_test.npy",allow_pickle=True)
    MatStadistics = np.load(path + "stadistics.npy")
    lb_tra[np.where(lb_tra == "sit")] = "stand"
    lb_test[np.where(lb_test == "sit")] = "stand"
    classes =["stand","walk","stairsup","stairsdown","bike"]
    datorealtra = (x_tra * np.reshape(MatStadistics[0:x_tra.shape[0], 1],
(x_tra.shape[0], 1))) + np.reshape(MatStadistics[0:x_tra.shape[0], 0],
(x_tra.shape[0], 1))
    datorealtest = (x_test * np.reshape(MatStadistics[0:x_test.shape[0], 1],
(x_test.shape[0], 1))) + np.reshape(MatStadistics[0:x_test.shape[0], 0],
(x_test.shape[0], 1))

    datorealtraplot=pcaplot.transform(datorealtra)
    datorealtestlot = pcaplot.transform(datorealtest)
    for model in vqmodel:

        PlotModelsVQ(x_tra, "e", vqmodel, "Full", configjson, pcaplot)
        xpre_tra, querrors_tra, Neuronwins_tra =
vqmodel[model].predict(x_tra)
        xpre_test, querrors_test, Neuronwins_test =
vqmodel[model].predict(x_test)



        datopretra = (xpre_tra *
np.reshape(MatStadistics[0:xpre_tra.shape[0], 1],(xpre_tra.shape[0], 1))) +
np.reshape(MatStadistics[0:xpre_tra.shape[0], 0],(xpre_tra.shape[0], 1))
        datopretest = (xpre_test *
np.reshape(MatStadistics[0:xpre_test.shape[0], 1],(xpre_test.shape[0], 1))) +
np.reshape(MatStadistics[0:xpre_test.shape[0], 0], (xpre_test.shape[0], 1))

        TableTrain = TrainStateTable(lb_tra, Neuronwins_tra,
vqmodel[model].neurons,classes)
        nombre="Efull "+ str(model)
        np.save('Trains/Tabla/tabla'+nombre+'.npy', TableTrain)

PlotReconstruccionError(datorealtraplot,pcaplot.transform(datopretra),querror
s_tra,model)

        PlotReconstruccionError(datorealtestlot,
pcaplot.transform(datopretest), querrors_test, model)
        predictions = PredictState(TableTrain,Neuronwins_tra)


        # Plot non-normalized confusion matrix
        confmat = confusion_matrix(lb_tra, predictions,
```

```
labels=["stand","walk","stairsup","stairsdown","bike"])

plot_confusion_matrix(confmat,["stand","walk","stairsup","stairsdown","bike"]
,title="confusion matrix"+str(model)+"train")

        predictions = PredictState(TableTrain, Neuronwins_test)
        # Plot non-normalized confusion matrix
        confmat = confusion_matrix(lb_test, predictions, labels=["stand",
"walk", "stairsup", "stairsdown", "bike"])
        plot_confusion_matrix(confmat, ["stand", "walk", "stairsup",
"stairsdown", "bike"],
                              title="confusion matrix" + str(model) + "test")

def PlotModelsVQ(x_tra, user, modelsMSCL, Composition, configjson, pcaplot):
    """
    Plot model_VQ
    :param x_tra: Data input train
    :param user: user
    :param modelsMSCL: pymscl object
    :param Composition: number of axis sensor
    :param configjson:json config file
    :param pcaplot:pca model object
    :return:
    """
    date =
datetime.datetime.fromtimestamp(time.time()).strftime('%Y%m%d_%H%M%S')
    Data = pcaplot.transform(x_tra)
    nModels = len(searchJson(configjson, ['Compress_model', 'MSCL',
'TargetFuntion']))
    Neurons = searchJson(configjson, ['Compress_model', 'MSCL', 'Neurons'])
    cont = 1
    j = 0
    fig, axs = plt.subplots(nModels + 1, 1, sharex='all', sharey='all')
    fig.tight_layout()
    axs[0].scatter(Data[:, 0], Data[:, 1],c=np.random.rand(3,))
    axs[0].set_title("Data")
    for i in modelsMSCL:
        centroids = pcaplot.transform(modelsMSCL[i].weight)
        if (nModels >= cont):
            axs[cont].scatter(centroids[:, 0], centroids[:,
1],c=np.random.rand(3,))
            axs[cont].set_title(i)
        if (nModels == cont):
            cont = 0
            plt.savefig(
                "Plots/Trainings/Graficamodelocolores/Compare_ModelsNeurons_"
+ str(user) + str(Composition) + str(Neurons[j]) + str(
                    date) + '.png')
            fig, axs = plt.subplots(nModels + 1, 1, sharex='all',
sharey='all')
            fig.tight_layout()
            axs[0].scatter(Data[:, 0], Data[:, 1])
            axs[0].set_title("Data")
            j += 1

        cont += 1


def PredictState (Table,neuronwinners):
```

```
    """
    Make neruon winner activity prediction
    :param Table: activity neuron Array 2d
    :param neuronwinners: array 2d
    :return: activity prediction
    """
    MatPredict = np.zeros((neuronwinners.shape[0],1)).astype(object)

    for row in range(len(neuronwinners)):

        MatPredict[row] = Table[neuronwinners[row]]


    return MatPredict


def TrainStateTable(trainlabel,neuronwinners,lentable,classes):
    """

    Create wininng activity neuron table
    :param trainlabel: Mat label array object
    :param neuronwinners: array neuronwinners
    :param lentable: Number of neurons
    :param classes: diferent classes
    :return: Table training
    """

    totalclass =len(classes)
    Table = np.zeros((lentable,totalclass),object)
    cont = 0
    for row in neuronwinners:
        if(trainlabel[cont] != "Notregistred"):
            col = classes.index(trainlabel[cont])
            Table[row[0],col] += 1
            cont += 1
    Table2 = np.zeros((lentable,1),object)
    for i in range(len(Table2)):

        winner=np.where(np.reshape(Table[i,:],(1,Table[i,:].shape[0])) ==
np.max(np.reshape(Table[i,:],(1,Table[i,:].shape[0])),axis=1))
        try:
            Table2[i] = classes[int(winner[1])]
        except:
            winners = winner[1]
            ranwin = np.random.choice(winners)
            Table2[i] = classes[int(ranwin)]

    return Table2

# Start Logg
logging.basicConfig(format='%(asctime)s - %(message)s', level=logging.INFO)

# Json with the information
configjson = "config"

test(configjson)
```

4. Implementación hardware

4.1. Upymscl

```
"""
-----------------------------------------------------

=====================================================
upythonMSCL algorithm
=====================================================

Autor: Jorge Vizarraga Turmo

Version:1.0

Function Reference
-----------------------------------------------------

_init()

random_weights(self, data_input)

random_weights_1(self, data_input)

initialize_train( self, data_length, inputs_number )

get_bmu_nbmu(self, row_data)

get_bmu(self, row_data)

get_lbmu(self, bmu, nmu)

refresh_Dw(self, data_input, datasample, lbmu, qerror_win, index_win, MatDW)

calculate_alpha(self,alpha,alphaini,alphafin,epoch)

adjust_weights(self, MatDW, alpha, index_win)

initialize_magnitudetemp()

update_magnitudetemp(self, mt, ma, bmu)

update_magnitude(self, mt, ma,target_function)

fit_model(self, X_train, X_validation, alphaini,
alphaend,target_function="unity")

predict(self, X_test)

"""

import math
import utime
import lib.array2upython as uap
import random

LIMIT_INITIAL_CENTROID = 1000
ALPHA_POWER = 0.5
Epsilon = 0.00001

class MSCL:
```

```
    """
    Parameters:
        Neurons:Total of vecocts to compone the codebook
        Bacth_size: number of data input to divide the dataset
        Number_batch: number of batch per epoch
ceil(TotalInputdata/Batch_size)
        Number of epoch: times to repeat the process
        Magnitude vector: Magnitudes to evaluate the second iteration between
BMU and NMU


    """

    def __init__(self, neurons=0, batch_size=0, tcycles=0,Composition =
"Full"):

        self.weight = list()
        self.neurons = neurons
        self.magnitude_vector = list()
        self.batch_size = batch_size
        self.tcycles = tcycles
        self.Composition = Composition

    def random_weights(self, data_input):
        """
        Create Random weights for Neurons,weight are selected from data_input

        ====================================
        Dimension = (Neurons*data_input_size)
        ====================================
        :param data_input: training data for take cols as weights (M)

        """
        self.weight = uap.matZeros( (self.neurons, data_input.cols()) )
        for i in range( len(self.weight) ):
            self.weight[i] = random.choice( data_input )

        self.magnitude_vector= uap.matUnos( (self.neurons,1) )

    def initialize_train( self, data_length, inputs_number ):
        """

        :param data_length: Number of data_samples use to train
        :param inputs_number: Number of data_inputs (cols) have the train
sample
        :return: vector qerrorwin ,indexwin and MatDW initialize to zeros
        """
        qerror_win = uap.matZeros((data_length, 1))
        index_win = uap.matZeros( (self.neurons, 1) )
        MatDW = uap.matZeros( (self.neurons, inputs_number) )

        return qerror_win, index_win, MatDW

    def get_bmu_nbmu(self, row_data):
        """

        :param row_data: Array 1D from DataInput
        :return: first [index,distance] and second BMU [index,distance]
```

```
        """

        qerrorlist = list()
        try:
            #Calculate Qerror
            data_input = uap.matUnos(self.neurons,row_data.rows())
            data_input = uap.matMult(data_input,row_data)
            qerrorlist = uap.pow2Mat(uap.matResta(( data_input , self.weight)
))

            #Search BMU and NMU
            indexbmu = qerrorlist.index(min(qerrorlist))
            bmu = [(indexbmu,qerrorlist[indexbmu])]
            qerrorlist.pop(indexbmu)
            indexnmu = qerrorlist.index(min(qerrorlist))

            #Update indexBMU if is after bmu index increase 1 (pop delete
indexbmu)
            if ( ( indexnmu >= indexbmu ) and ( indexnmu <= (len( qerrorlist
) - 1 ) ) ):
                nmu = [( indexnmu + 1, qerrorlist [indexnmu]) ]
            else:
                nmu = [( indexnmu , qerrorlist [indexnmu] )]

        except Exception as e:
            print(e, exc_info=True)

        return bmu,nmu

    def get_bmu(self, row_data):
        """

        :param row_data: Array 1D from DataInput
        :return: first [index,distance]
        """

        qerrorlist = list()

        try:
            #Calculate Qerror
            data_input = uap.matUnos(self.neurons,row_data.rows())
            data_input = uap.matMult(data_input,row_data)
            qerrorlist = uap.pow2Mat(uap.matResta(( data_input , self.weight)
))
            # BMU index
            bmu = qerrorlist.index( min(qerrorlist))

        except Exception as e:
            print(e, exc_info=True)

        return [(bmu, qerrorlist[bmu])]

    def get_lbmu(self, bmu, nmu):
        """

        :param bmu: Best neuron for the datainput sample
        :param nmu: Second neuron for the datainput sample
        :return: Lbmu the winner of qerror * magnitude_vector
[(index,querror,querror*M)]
```

```
        """
        qerrorM = list()
        indexlbmu = [bmu[0][0], nmu[0][0]]
        qerrorM.append(  (bmu[0][1] * self.magnitude_vector[bmu[0][0]] ) )
        qerrorM.append( (nmu[0][1] * self.magnitude_vector[nmu[0][0]] ))
        lbmu = qerrorM.index( min( qerrorM ) )

        return [( indexlbmu[lbmu], qerrorM[lbmu] )]

    def refresh_Dw(self, data_input, datasample, lbmu, qerror_win, index_win,
MatDW):
        """

        :param lbmu: Winner neuron
        :param index_win: vector for winning index for data_inputs
        :param qerror_win: vector for qerror winners for data_inputs
        :param MatDW: Mat to acumulate qerror and weights
        :return: Updated index_win, qerror_win, MatDW
        """
        qerror_win[datasample] = lbmu[0][1]
        index_win[lbmu[0][0]] += 1

        MatDW[lbmu[0][0]] += ( uap.matResta(data_input ,
self.weight[lbmu[0][0]] ))

        return qerror_win, index_win, MatDW

    def calculate_alpha(self,alpha,alphaini,alphafin,epoch):
        """
        Calculate the learning factor forced to decay with iteration time
        :param alpha: Actual value for alpha (start in alphaini)
        :param alphaini: constant value for alpha in t=0
        :param alphafin: constant valua for alpha expected in t=T (number of
epoch)
        :param epoch: Number of epoch t
        :return: alpha (t)
        """
        try:
            #alpha highstart
            #alpha = math.pow( ( alphafin * ( alphaini / alpha) ), (epoch /
self.tcycles) )

            #alpha softstart
            alpha = (alphaini + (alphafin - alphaini) * math.pow((epoch /
self.tcycles), ALPHA_POWER))

        except Exception as e:
            print( e, exc_info=True)

        return alpha

    def adjust_weights(self, MatDW, alpha, index_win):
        """
        Adjust weights for neurons to next cycle follow this ecuation:

        ======================================================
        Weights(t+1)=weights(t)+alpha*(data_iputs - weights_Lbmu)
        ======================================================
```

```
        :param MatDW: Codebook with acumulation (data_input-weightLBMU)
        :param alpha: learning factor calculate for t epoch
        :return:
        """
        try:
            index_nondivisible = index_win + Epsilon # +Epsilon ZERO division
            AcumulationSum = ( alpha * ( uap.matDiv(MatDW ,
(index_nondivisible) )))
            self.weight = uap.matSum(self.weight, AcumulationSum)


        except Exception as e:
            print( e, exc_info=True)


    def initialize_magnitudetemp(self):
        """
        Inicializar magnitude temp vector to evalue next magnitude
        :return: Mt , Ma vector zeros magnitude_vector size
        """
        M_size = len( self.magnitude_vector )
        Mt = uap.matZeros( (M_size,1) )
        Ma = uap.matZeros( (M_size,1) )

        return Mt, Ma

    def update_magnitudetemp(self, mt, ma, bmu):
        """
        We accumulate in the Mt vectors the value of the BMU Querror and Ma
an activation for the BMU index

        :param mt: Accumulate Querror of BMU'S
        :param ma: Number of winning for this neurons
        :param bmu: Winner neuron
        :return:
        """
        mt[bmu[0][0]] += bmu[0][1]
        ma[bmu[0][0]] += 1

        return mt,ma


    def update_magnitude(self, mt, ma,target_function):
        """
        Calculate new magnitude vector depend of target_function
        :param mt: BMU Magnitude temporal
        :param ma: BMU Activation temporal
        :param target_function:
        :return: Update vector M value for next epoch
        """
        M_size = len(self.magnitude_vector)
        magnitude = uap.matZeros((M_size, 1))

        if target_function == "unity" :
            magnitude = self.magnitude_vector

        if target_function == "FSCL" :
            magnitude = ma
```

```
        if target_function == "ESCL":

           # for i in range(M_size):

            #    if ma[i] != 0 :
            magnitude = uap.matDiv(mt , (uap.matSum(ma , Epsilon))) #
+Epsilon ZERO division

        return magnitude

    def fit_model(self, X_train, X_validation, alphaini,
alphaend,tarjet_function="unity"):
        """
         Fit model with the paramaters

        :param X_train: Train data Array 2d
        :param X_validation: Validation data Array 2d
        :param alphaini: Learning factor ini
        :param alphaend: Learning factor end
        :return: model fit
        """
        cont_ciclos = 0
        cont_batches = 0
        cont_datos = 0
        alpha = alphaini
        weights=uap.matZeros((self.neurons,1))

        self.random_weights(X_train)

        while cont_ciclos != self.tcycles:
            #Init conts
            cont_batches = 0
            cont_datos = 0
            #Calculate weights
            qerror_win, index_win, MatDW = self.initialize_train(
X_train.rows(), X_train.cols() )
            Tb = ( math.ceil( X_train.rows() / self.batch_size ) )

            while cont_batches != Tb :

                while (cont_datos != self.batch_size) and (cont_datos + (
cont_batches * self.batch_size ) <= (X_train.rows())-1) :
                    data_sample = cont_datos + ( cont_batches *
self.batch_size )
                    bmu,nmu = self.get_bmu_nbmu( X_train[data_sample] )
                    lbmu = self.get_lbmu( bmu, nmu )
                    qerror_win, index_win, MatDW = self.refresh_Dw(
X_train[data_sample], data_sample, lbmu, qerror_win, index_win, MatDW)
                    cont_datos += 1

                cont_batches+=1
                cont_datos=0

            cont_batches = 0
            cont_datos = 0
            cont_ciclos += 1


            #Update weights
```

```
            alpha = self.calculate_alpha( alpha, alphaini, alphaend,
cont_ciclos )

            self.adjust_weights( MatDW, alpha, index_win )

            #Start Magnitude update
            Mt, Ma = self.initialize_magnitudetemp()
            while cont_batches != Tb:
                while (cont_datos != self.batch_size) and (cont_datos + (
cont_batches * self.batch_size ) <= (X_train.rows())-1)  :
                    data_sample = cont_datos + ( cont_batches *
self.batch_size)
                    bmu= self.get_bmu(X_train[data_sample])
                    Mt, Ma = self.update_magnitudetemp( Mt, Ma, bmu)
                    cont_datos += 1
                cont_batches += 1
                cont_datos = 0


            #update Magnitude per epoch
            self.magnitude_vector =
self.update_magnitude(Mt,Ma,tarjet_function)



    def predict(self, X_test):
        """
        Return predicted values from input data 2d
        :param X_test: Test data array 2d
        :return:
        """

        cont_datos = 0
        qerror_winner = uap.matZeros( ( X_test.rows(), 1) )
        index_winner = uap.matZeros( (X_test.rows(), 1) )
        Matpredict = uap.matZeros( (X_test.rows(), self.weight.cols()) )
        while cont_datos != X_test.shape[0]:
            data_sample = cont_datos
            bmu = self.get_bmu( X_test[data_sample] )
            index_winner[data_sample] = bmu[0][0]
            qerror_winner[data_sample] = bmu[0][1]
            Matpredict[data_sample] =  self.weight[bmu[0][0]]
            cont_datos += 1



        return Matpredict, qerror_winner, index_winner
```

4.2. Main

```
import machine
import utime
import lib.upymscl
import lib.bno055
import lib.memo
import lib.array2upython as uap
import _thread
from machine import UART
import lib.influx
```

```
Queue = list()
batch_size = 200
delay = 0.005
fs = 50
T = 1
neurons = 1024
batch_size = 200
tcycles = 20
Composition = "Full"
namefile ="training"

def ini_i2C():

    i2c = machine.I2C(1)

    return i2c
def ini_uart():
    global uart
    uart = UART(0, baudrate=115200)
    uart.init(115200, bits=8, parity=None, stop=1)

def synctime ():
    global rtc,uart
    uart.write("Syncro RTC ...\n")
    rtc = machine.RTC()
    rtc.ntp_sync("pool.ntp.org")
    utime.sleep(0.750)
    uart.write(str(rtc.synced())+"\n")
    uart.write(str(utime.time())+"\n")

def th_func(delay, id,imu):
    global Queue
    uart.write("Thread Start\n")
    while True:
        utime.sleep(delay)
        Queue.append(imu.accel())

def initsensor(i2c):

    imu = bno055.BNO055(i2c)
    calibrated = False
    while not calibrated:
        utime.sleep(1)
        if not calibrated:
            calibrated = imu.calibrated()
            print('Calibration required: sys {} gyro {} accel {} mag
{}'.format(*imu.cal_status()))
        uart.write('Temperature {}°C'.format(imu.temperature()))
        uart.write('Mag       x {:5.0f}    y {:5.0f}     z
{:5.0f}'.format(*imu.mag()))
        uart.write('Gyro      x {:5.0f}    y {:5.0f}     z
{:5.0f}'.format(*imu.gyro()))
        uart.write('Accel     x {:5.1f}    y {:5.1f}     z
{:5.1f}'.format(*imu.accel()))
        uart.write('Lin acc.  x {:5.1f}    y {:5.1f}     z
{:5.1f}'.format(*imu.lin_acc()))
        uart.write('Gravity   x {:5.1f}    y {:5.1f}     z
{:5.1f}'.format(*imu.gravity()))
        uart.write('Heading     {:4.0f} roll {:4.0f} pitch
```

```
{:4.0f}'.format(*imu.euler()))

    return imu

def startACCdata(imu,delay,idthread):

    synctime()
    _thread.start_new_thread(th_func, (delay, id,imu))

def Sensorcompress(windowsize,centroid):
    win= list()
    model = upymscl.MSCL()
    model.weight = centroid
    while(1):
        if(len(Queue)!=0):
            dato = Queue.pop(0)
            win.append(dato)
            if(len(win) == windowsize):
                winmean = uap.matMean(win)
                winstd = uap.matStd(win)
                win = uap.matDiv((uap.matResta(win,winmean)),winstd)
                compressdata = model.predict(win)
                influx.sendata(compressdata,winmean,winstd)



def Sensortrain(windowsize,tf,alphaini,
alphafin,neurons,batch_size,tcycles,Composition):
    win=list()
    FinTrain = False
    batch = list()
    model = upymscl.MSCL(neurons,batch_size,tcycles,Composition)
    while(not FinTrain):
        if(len(Queue)!=0):
            dato = Queue.pop(0)
            win.append(dato)
            if(len(win) == windowsize):
                winmean = uap.matMean(win)
                winstd = uap.matStd(win)
                win = uap.matDiv((uap.matResta(win,winmean)),winstd)
                batch.append(win)
                if(len(batch)==batch_size):
                    model.fit(win)
                    FinTrain = True

    return model


"""
=============================================
MAIN
=============================================
"""
#Start sensor comunication
ini_uart()
i2c = ini_i2C()

#initSensor
imu = initsensor(i2c)
```

```
#StartSensorData
startACCdata(imu,delay,0)

winsize = fs * T
Recordtrain = memo.exitfile(namefile)

if(RecordTrain):
    centroid = memo.restorecentroid(namefile)
    Sensorcompress(windowsize,centroid)

else:
    model=Sensortrain(windowsize,tf,alphaini,
alphafin,neurons,batch_size,tcycles,Composition))
    memo.createfil(namefile)
    memo.savecentroid(namefile,model.weight)
    machine.reset()
```